VŠB – TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky

Diplomová práce

2011 Marcel Fajkus

VŠB – TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Návrh MATLAB serveru katedry telekomunikační techniky

Design of the MATLAB server for Department of Telecommunications

2011 Marcel Fajkus

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Marcel Fajkus**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Návrh MATLAB serveru katedry telekomunikační techniky
Design of the MATLAB server for Department of Telecommunications

Zásady pro vypracování:

1. Popis MATLAB serveru.
2. Popis zprovoznění MATLAB serveru na web serveru APACHE.
3. Popis zprovoznění MATLAB serveru na web serveru APACHE v případě, že na serveru není instalován MATLAB.
4. Vytvoření vzorové aplikace na MATLAB serveru.

Seznam doporučené odborné literatury:

ZAPLATÍLEK, K.; DOŇAR, B. *MATLAB: tvorba uživatelských aplikací.* Vydání 1. Praha, BEN - technická literatura, 2004. 215 s. ISBN 80-7300-133-0.

Dále dle doporučení vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 19.11.2010

Datum odevzdání: 06.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

prof. RNDr. Vladimír Vašinek, CSc.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

„Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.“

V Ostravě ……. Podpis

Poděkování
Na tomto místě bych chtěl poděkovat vedoucímu své diplomové práce
Ing. Janu Skapovy, Ph.D. za rady a připomínky k obsahu a formě zpracování.

Abstrakt

Diplomová práce se zabývá využitím výkonového jádra MATLABu na Internetu. V práci jsou popsány způsoby jak zprovoznit Matlab Web Server verze 2006a a nižší a jak nasadit MATLAB aplikace jako webové aplikace na webový server s využitím MATLABu verze 2006b a vyšší. Jsou popsány jednotlivé fáze vývoje webové aplikace a v závěru jsou vytvořeny dvě ukázkové aplikace.

Klíčová slova

MATLAB, MATLAB Compiler, MATLAB Builder JA, WebFigure, MWArray, servlet, applet

Abstract

This thesis deals with using power of core of MATLAB application on the Internet. The paper describes ways, how ti implement the Matlab Server versions 2006a and lower, and how to deploy MATLAB applications as web applications on the Webserver using MATLAB 2006b version or higher. There are described individual phases of development of web applications and created two sample applications in conclusion.

Key words

MATLAB, MATLAB Compiler, MATLAB Builder JA, WebFigure, MWArray, servlet, applet

1. Úvod

Programovací jazyk MATLAB je určen pro vědeckotechnické účely, simulace, paralélní výpočty, modelování, návrh algoritmů, analýzu a zpracování dat, měžení a zpracování signálu apod. Využití je obrovské a z tohoto důvodu bylo snahou této diplomové práce zpřístupnit jeho sílu uživatelům pomocí Internetu. Aplikace vytvořená v MATLABu se převede na Java komponentu, kterou lze pak nasadit na webový/aplikační server, kde se spouští a provádí požadované výpočty.

Ve druhé kapitole je popsán koncept Matlab Web Serveru, který byl součástí MATLABu verze R2006a a nižší. Ukazuje jak vytvářet MATLAB aplikace, které se spouští v MATLABu na webovém serveru. Třetí kapitola popisuje koncept MATLABu verze R2006b a vyšší. Seznamuje čtenáře s jednotlivými nástroji MATLABu, které převádí MATLAB aplikace do jiných jazyků jako je Java, C nebo C++.

Čtvrtá kapitola se zabývá teorií Javy. Java je potřebná pro nasazení MATLAB aplikací na web a proto je zde uvedeno, jakým způsobem Javu nainstalovat, jak nastavit systémové proměnné a jak pracovat s nástroji Javy, bez kterých se při vývoji neobejdeme.

Stěžejní částí diplomové práce je kapitola pátá, ve které je ukázáno, jakým způsobem se převádí MATLAB aplikace na Java komponentu, kterou lze následně využít v Java desktopové aplikaci anebo nasadit na web a využít na Internetu.

Spouštění Java komponenty na Internetu vyžaduje kromě webového serveru také aplikační server, který umožňuje spouštět Javovské programy na straně serveru. Šestá kapitola se tedy zabývá aplikačním serverem, jeho instalací a konfiguraci. V sedmé kapitole jsou uvedeny výhody používání appletů jako uživatelského rozhraní Matlab aplikace oproti klasickým HTML formulářům. Dále jsou uvedeny principy komunikace appletu se servletem na straně serveru. MATLAB aplikace v podobě Java komponenty jsou spouštěny na straně serveru právě pomocí servletů. Osmá kapitola se zabývá vytvářením servletů a jejich nasazením na server.

Devátá kapitola popisuje použití absraktní třídy MWArray, která slouží pro převod datových typů parametrů při spouštění MATALB aplikace. MATLAB aplikace umístěná v Java komponentě je spouštěná servletem. Parametry předávané Java komponentě musí být datového typu některé z potomků této abstraktní třídy, které slouží jako prostředník mezi Javou a MATLABem. Zde jsou pospány pravidla pro převod datových typů.

V desáté kapitole je rozebrána třída WebFigures, která se využívá pro manipulaci s grafikou vygenerovanou MATLABem.

V jedenácté kapitole je ukázáno, jakým způsobem se vyvtáří webová aplikace využívající MATLAB aplikaci a jak jí nasadit na aplikační server. Ve dvanácté kapitole jsou umístěny ukázky dvou vytvořených aplikací a uživatelské rozhraní matlab serveru pro spouštění aplikací.

.

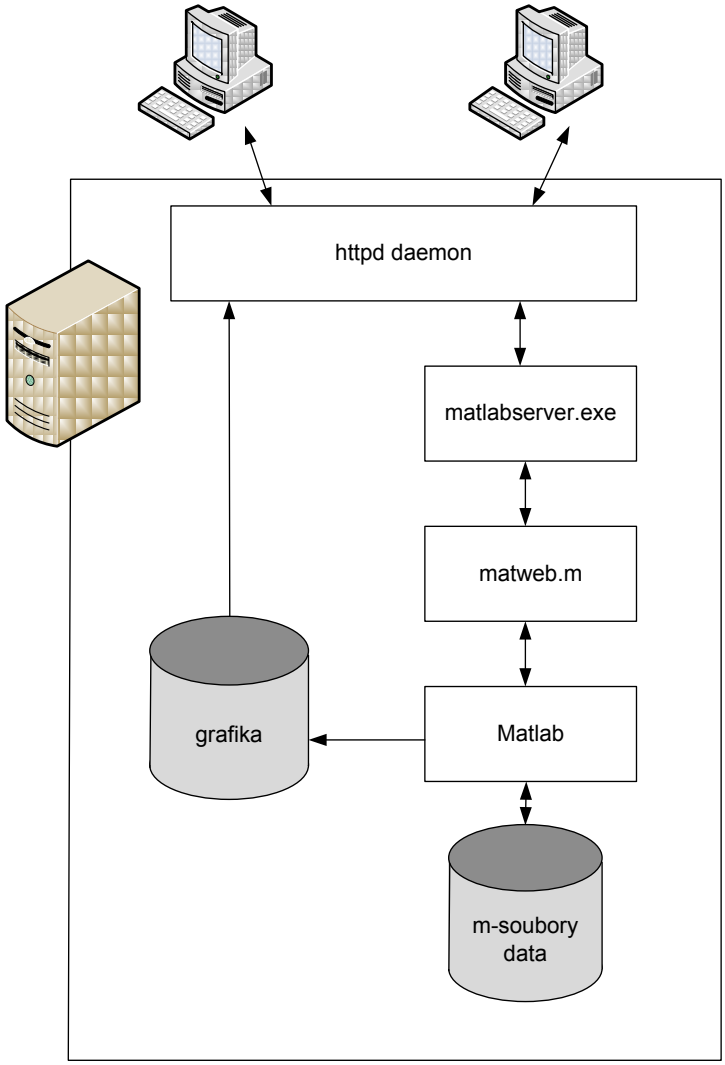
2. Matlab Web Server

Matlab Web Server je toolbox Matlabu, který je součástí Matlabu do verze R2006a. Matlab Web Server umožňuje využít matematického programu Matlab po síti. Matlab aplikaci uloženou na jednom počítači připojeného k Internetu lze spustit na kterémkoliv počítači s webovým prohlížečem a připojením na Internet.

Matlab Web server je tvořen cgi scriptem pro extrakci vstupních dat z formuláře, vícevláknovým démonem, který přijímá požadavky na spuštění aplikace a soubor `matweb.m`, který spustí požadovaný m-file na serveru. Pro provoz jsou dále potřebné 2 konfigurační soubory a vstupní/výstupní html stránky.

2.1. Princip komunikace

Klient s přístupem na Internet si otevře html stránku aplikace. Do formuláře vyplní parametry potřebné pro spuštění aplikace a odešle požadavek pro zpracování na webový server. Na webovém serveru požadavek přijme skript CGI `matweb.exe`. Tento skript je součástí Matlabu a je nutné jej nakopírovat do adresáře CGI-BIN na webovém serveru. Skript `matweb.exe` přijme data z formuláře a odešle je démonu `matlabserver.exe`. Tento démon je služba, která je spuštěna a čeká na požadavky odeslanými klientem `matweb.exe`. Po přijetí požadavku spustí program `matweb.m`, kterému předá přijatá data. Tento program následně spustí požadovaný m-file, který zpracuje úlohu a vypočítá výsledky. Tyto výsledky se potom naimportují do připravené šablony HTML, která se odešle nazpět webovému prohlížeči. Na obrázku 1. je blokové schéma Matlab Seb serveru.



Obr. 1: Schéma Matlab Web serveru

2.2. Vytvoření aplikace pro Matlab Web Server

Matlab Web Server aplikace jsou kombinací M-souborů, značkovacího jazyka HTML a grafiky. Vývoj aplikace vyžaduje následující kroky:

- vytvoření HTML dokumentu pro sběr vstupních dat zadaných uživatelem, například pomocí formuláře,
- vytvoření konfiguračního souboru,
- vytvoření m-souboru, který,
    - přijme údaje zadané v HTML dokumentu,
    - analyzuje data a vytvoří požadovanou grafiku nebo textové výsledky,
    - umístí výstupní data do MATLAB struktury,
    - výslednou strukturu dat uloží do výstupní šablony HTML dokumentu.

2.3. Požadavky na Matlab Web Server

Matlab Web Server má stejné požadavky na hardware i software jako Matlab až na velikost paměti. Nároky na pamět se liší podle počtu spuštěných Matlab programů. Každý běžící program v Matlabu vyžaduje alespoň 256 KB paměti.

Dále je potřeba webový server např. Apache, který běží na stejném stroji jako Matlab Web Server nebo na stroji, který má přístup k MWS po síti. Webový server musí podporovat spouštění skriptů CGI (Common Gateway Interface).

2.4. Součásti Matlab Web Serveru

Matlab Web Server se skládá z těchto částí.

Program `matlabserver` je vícevláknový TCP/IP server. Řídí komunikaci webové aplikace s Matlabem.

Konfigurační soubor `matlabserver.conf` slouží ke konfiguraci matlabserveru. Lze nastavit číslo portu na kterém poslouchá matlabserver nebo počet současně spuštěných Matlab aplikací.

Program `matweb.exe` využívá cgi skripty k extrahování dat z HTML dokumentu a předává je serveru `matlabwerver.exe`. Do formuláře v HTML dokumentu se vloží skryté pole s názvem *mlmfile* a hodnotou, která určuje názvem m-souboru, který se má spustit.

Program `matweb.m` je m-soubor, který je spuštěn démonem matlabserver. Následně tento program zavolá požadovaný m-file, který provede výpočty a sestaví grafiku.

Konfigurační soubor `matweb.conf` je zapotřebí ke spojení se serverem `matlabserver.exe`. V tomto souboru se konfigurují všechny aplikace. Zapisují se zde názvy všech programů a jejich argumentů, dále názvy souborů pro logování, informace o adresářích apod.

Soubor `hosts.conf` definuje, které počítače se mohou k Matlab Web Serveru připojit.

2.5. Aplikace Matlab Web Serveru

V této části si ukážeme vytvoření jednoduché aplikace. Aplikace vypočte spektrum signálu (sinusový nebo kosinusový) na základě zadaných parametrů jako je vzorkovací kmitočet, kmitočet signálu a amplituda signálu. Vypočtené spektrum zobrazí v grafu, který se vloží do připraveného HTML souboru. Tento HTML soubor se odešle nazpět klientovi.

2.5.1. Vstupní HTML soubor

Takto vypadá formulář pro sběr dat potřebných pro simulaci. Data se odešlou na CGI skript `matweb.exe`.

```
<form action="/cgi-bin/matweb.exe" method="POST">
  <input type="hidden" name="mlmfile" value="fourtransf">
  <table>

  <label>Signal</label>
  <p><select name="funkce" style="width: 100px;">
    <option value="sin">sinus</option>
    <option value="cos">kosinus</option>
  </select></p>

  <label>Vzorkovaci kmitocet</label>
  <p><input type="text" name="fvz" style="width: 100px;"></p>

  <label>Frekvence</label>
  <p><input type="text" name="f" style="width: 100px;"></p>

  <label>Amplituda</label>
  <p><input type="text" name="amplituda" style="width: 100px;"></p>

  <p><input type="submit" name="Submit" value="Submit"></p>
</form>
```

2.5.2. fourtrans.m

Zde je výpis m-souboru, který přečte vstupní data, provede výpočty a výsledky vloží do připraveného HTML souboru. Kód je rozdělen do několika kroků pro snadnější orientaci.

V prvním kroku se definuje výstupní proměnná `retstr`, jež je uvedena v deklaraci funkce. V druhém kroku se nastaví pracovní adresář dané aplikace. Tyto informace jsou uvedeny v konfiguračním souboru aplikace. Ve třetím kroku se získají vstupní parametry. Jsou zde ošetřeny případy, kdy nějaká hodnota ve formuláři nebyla vyplněna. V takovém případě se použije defaultní hodnota. Čtvrtý krok představuje samotný výpočet fourierovy transformace a vytvoří se grafika, která se uloží jako obrázek do pracovního adresáře. Následuje uložení výstupních dat do výstupní datové struktury. V šestém kroku se výstupní datová struktura vloží do výstupního HTML souboru.

Obsah souboru je rozsáhlý, a proto je umístěn v příloze na CD

`priloha/zdrojove_kody/fourtrans.m`.

2.5.3. Výstupní HTML soubor

Ve výstupním HTML souboru se umístí obrázek, jež byl vytvořen Matlabem v m-filu naší aplikace.

```
<img border=0 src="/aplikace/fourtransf/$filename$"
     width="530px" height="384px;" >
```

2.5.4. matweb.conf

V konfiguračním souboru přidáme položku naší aplikace s názvem *fourtransf*. Zde je potřeba nadefinovat pracovní adresář aplikace.

```
[fourtransf]
mlserver=Marcel
mldir=C:\Program Files\MATLAB71/toolbox/webserver/wsdemos
```

2.6. Výhody a nevýhody

Mezi výhody této koncepce patří jednoduchost vytvoření aplikace. Naopak k hlavnímu nedostatku této koncepce se řadí povinnost mít zakoupený a nainstalovaný Matlab.

3. Matlab Server R2006b

Matlab server Matlabu verze R2006b a výše představuje úplně jinou koncepci webového serveru oproti starší koncepci starších verzí Matlabu, konkrétně Matlab Web Serveru. Pojmenování Matlab serveru v tomto případě není zcela vhodné, protože se o žádný server prakticky nejedná. Matlab verze 2006b a výše vychází z toolboxů, které vytvoří z m-filu kód v programovacím jazyce Java nebo C, které pak lze využít v Java aplikacích, aplikacích založených na .NET apod. Například vytvořený kód v Javě lze spoustět servletem na webovém serveru. Tento servlet pak komunikuje s vytvořenou Java aplikací a zobrazuje výsledky. Uživatel si pak pouze spustí aplikaci v podobě servletu, který zařídí výpočet a zobrazení dat.

Jedním z nedostatků u Matlab Web Serveru bylo to, že pro spuštění aplikace musel být samotný Matlab nainstalovaný na serveru. Anebo musel být alespoň přístupný po síti, což v tomto novém přístupu odpadá.

V této diplomové práci se zabývám využitím Matlabu na webu, a proto zde bude popsáno pouze vytváření Java komponent pomocí Matlab Builder JA. Kromě tohoto nástroje existuje ještě Matlab Builder EX a Matlab Builder NE.

3.1. Požadované produkty

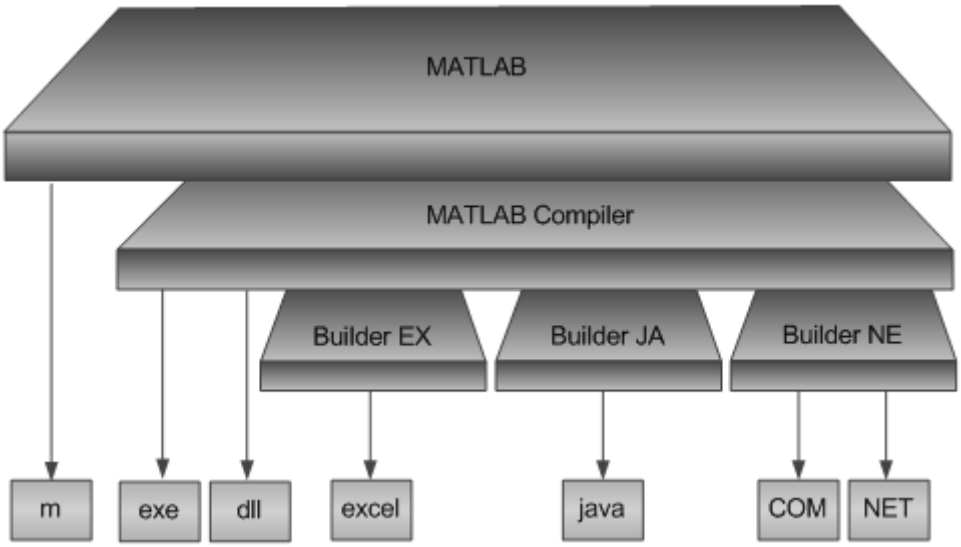
Pro vývoj Java komponent je potřeba mít nainstalované následující nástroje:

- Matlab,
- Matlab Compiler (překladač),
- Matlab Builder JA.

Pro samotné spuštění Java aplikací je potřeba mít nainstalované běhové prostředí :

- Matlab Component Runtime (MCR).

Matlab je programové prostředí a skriptovací jazyk pro vědeckotechnické a numerické výpočty, simulace, analýzy a prezentace dat. Samotný Matlab je potřeba pro naprogramování Matlab aplikace a pro její otestování. Následně pomocí Matlabu a s ostatními nástroji se vytváří Java komponenty, .NET komponenty, objekty COM apod. Schéma nástrojů pro tvorbu komponent je na následujícím obrázku. Popsány jsou v následujících kapitolách.



Obr. 2: Koncepce Matlabu pro vývoj komponent

3.2. Matlab Builder JA

Matlab Builder umožňuje vytvářet Java třídy z MATLAB souborů, nazývaných m-fily. Tyto Java třídy mohou být integrovány do programů v jazyce Java a nasazeny na stolní počítače jako desktopové aplikace nebo na webové servery dostupné ze servletů.

Takto vygenerovaný Java kód lze spustit na všech platformách podporovaných Matlabem.

3.3. Matlab Compiler

Matlab Compiler je nástroj, který spolupracuje s Matlab toolboxy a umožňuje vytvářet komponenty, jež mohou komunikovat s externími aplikacemi. Můžeme vytvářet komponenty EX pomocí Matlab Bulder EX, které lze využívat v programech Microsoft Excel. Matlab Builder NE rozšiřuje Matlab Compiler o schopnost vytvářet komponenty .NET a objekty COM.

V této diplomové práci se budu zabývat pouze Matlab Builder JA, který vytváří Java komponenty, které budeme nasazovat do webových apletů. Kromě Java komponent umí Matlab Compiler ve spolupráci s Matlab Builder NE vytvářet také knihovny C a C++ jako dynamicky linkované knihovny nebo knihovny DLL na operačních systémech Windows. Navíc umí také vytvářet z m-filů ve spolupráci s dalšími toolboxy také spustitelné aplikace EXE běžících na platformách UNIX, Windows a Macintosh.

3.3.1. Úkoly Matlab Compileru při vytváření aplikace

Při vytváření aplikace či komponenty, Matlab Compiler provádí tyto úkoly.

1. Zpracovává a klasifikuje požadavek a jeho argumenty podle typu poskytovaných souborů.

2. Analyzuje soubory pro závislosti. Závislosti ovlivňují rozmístění a vznikají z funkcí, které jsou volány ze souboru. Rozmístění souborů je ovlivněno:
    a. typem souboru – Matlab, Java, MEX, JAR a atd.,
    b. umístěním souboru – Matlab, Matlab toolbox, uživatelský kód a atd.,
    c. rozmístění souboru – zda je soubor nasazen mimo Matlab,
3. Ověřuje MEX soubory, obzvlášť jsou ověřeny vstupy MEX funkcí.,
4. Vytvoří CTF archiv ze vstupních souborů.
5. Generuje cílový specifický obal kódu, například jazyk C požaduje jiný obal než jazyk Java.
6. Vyvolá kompilaci v součinnosti Matlab Builder JA k vytvoření binární komponenty, spustitelné aplikace, JAR souboru nebo jiných typů.

Více informací o způsobu činnosti Matlab Compileru najdete na stránkách Mathworks [3].

3.4. Matlab Component Runtime (MCR)

Důležitou součásti při vývoji komponent je běhové prostředí MCR (Matlab Component Runtime), které představuje výpočtové jádro Matlabu a poskytuje sdílené knihovny Matlabu pro vytvořené komponenty již zmiňovaných technologií. MCR umožňuje spouštět vytvořené komponenty na počítačích, kde není nainstalován Matlab. Podmínkou je, aby na počítači bylo nainstalováno právě toto běhové prostředí dřív, než se pokusíme spustit vytvořenou aplikaci využívající komponenty vytvořené pomocí Matlab Builder JA. Toto je výhoda oproti koncepci Matlab Web serveru R2006a a starších, kde pro spuštění aplikace musel být Matlab na počítači nainstalován anebo alespoň přístupný po síti.

3.5. Systémové požadavky

Následuje definice systémových požadavků pro verzi Matlab R2010a, která bude nasazena pro Matlab server katedry telekomunikační techniky Vysoké školy Báňské, technické univerzity Ostrava.

Windows

Požaduje operační systém alespoň Windows XP se Service Packem 3 a novější včetně Microsoft Windows 7. Dále požaduje procesor Intel nebo AMD architektury x86 podporující instrukční sadu SSE2. Na disku alespoň 1 GB pouze pro MCR a 3 až 4 GB pro typickou instalaci. Minimální požadavek na operační paměť RAM je 1024 MB, doporučená hodnota 2048 MB.

UNIX

Instalace Matlabu R2010a podporuje následující Linuxové disrtibuce

- Ubuntu 8.04, 8.10, 9.04 a 9.10,
- Red Hat Enterprise Linux 5.x,
- SUSE Linux Enterprise Desktop 11.x,
- Debian 5.x.

U těchto distribucí je zaručeno, že MATLAB bude fungovat. U ostatních distribucí s jádrem 2.6 a vyšším a s glibc verze 2.5 a vyšší by mělo také fungovat. To už ale není ověřeno. Požadavky na procesor, velikost disku a RAM jsou stejné jako u platformy Windows.

3.6. Instalace na OS Windows

Samotná instalace není nikterak náročná a představuje pouze zvolení požadovaných nástrojů, které se mají nainstalovat. V tomto bodě je potřeba zvolit Matlab Builder JA a Matlab Compiler.

3.7. Instalace na OS Linux

Instalace na operační systém Linux již není takovým standardem jako instalace na Windows. Proto následuje postup instalace na OS Linux distribuce Ubuntu.

ISO soubor `R2010a_UNIX.iso` s Matlabem nakopírujeme do adresáče `/home/user/matlab`. Do tohoto adresáře také nakopírujeme licenci, která je umístěna v souboru `license.dat`.

Následně připojíme obraz do adresáře `/mnt/matlab`

```
mkdir /mnt/matlab
mount /home/user/matlab/R2010a_UNIX.iso –o loop –t iso9660 /mnt/matlab
```

Vytvoříme cílový adresář, kam chceme Matlab nainstalovat

```
mkdir /usr/local/matlab
```

a spustíme instalaci

```
/mnt/matlab/install* -t
```

Následně nás instalátor vyzve k zadání licenčního souboru a k určení cílového adresáře pro instalaci Matlabu.

3.8. Vývoj komponenty

Vytvoření komponenty, kterou chceme spustit ve webovém serveru je potřeba rozdělit do několika částí, které se rozdělí na více vývojářů. Vytvořit takovou komponentu totiž vyžaduje znalosti nejen skriptovacího jazyku Matlab, ale také programovacího jazyka Java. Co je tedy potřeba provést v jednotlivých částech vývoje je popsáno níže.

- Matlab Programátor
    - Musí porozumět potřebám koncového uživatele a vytvořit matematické modely.
    - Napíše Matlab kód, jež řeší požadovanou funkci aplikace.
    - Vytvoří spustitelný kód (obvykle s podporou Java programátora).
    - Vytvoří balíček komponent (soubor JAR) pro distribuci koncovým uživatelům.
    - Předá komponenty Java programátoru.
- Java programátor
    - Napíše Java kód pro spuštění balíčku vytvořeného Matlab programátorem.
    - Integruje komponenty do uživatelského prostředí.
    - Použije komponenty v podnikové Java aplikaci nebo v servletu pro spuštění ve webovém prohlížeči.
    - Otestuje spustitelnost Java aplikace v koncovém uživatelském prostředí.
- Externí uživatel
    - Používá a testuje řešení vytvořené Matlab a Java programátorem.

Vývoj Java komponenty je popsán v kapitole 5 a vývoj webové aplikace využívající Java komponentu je popsán v kapitole 11.

3.8.1. Co je potřeba znát

Matlab programátor musí znát práci s buňkami pole a strukturami. Java programátor musí znát jazyk Java, objektově orientované programování, dále musí umět vytvářet servlety, používat síťovou komunikaci a v neposlední řadě se musí seznámit a používat abstraktní třídu MWArray, která se stará o konverzi datových typů při komunikaci Matlabu a Javy. Práce s touto třídou je popsána v kapitole 9.

4. Java

Programovací jazyk Java, vytvořený firmou SUN, patří k těm nejpoužívanějším, protože umožňuje přenositelnost mezi různými platformami. To znamená, že zdrojový kód není závislý na operačním systému ani na hardwaru počítače. Toto je zajištěno kompilací. Zdrojové kódy jsou kompilovány do tzv. java bytecodu, který ještě stále není závislý na SW a HW počítače. Při spuštění Java programu se tento zkompilovaný kód teprve převede na strojový kód daného procesoru s ohledem na operační systém. Převedení na strojový kód zajišťuje JVM (Java Virtual Machine).

JVM je dostupný pro většinu platforem a proto stačí program napsat jen jednou, zkompilovat a následně spustit na jakékoliv platformě. JVM je dodáván společně se sadou standardních knihoven API (Application Programming Interface). JVM a API tvoří celek a je poskytován jako JRE (Java Runtime Environment).

Pro překládání (též kompilaci) programů Java je potřeba překladač nebo-li kompilátor. Kompilátor je součástí JDK (Java Development Kit). JDK také navíc obsahuje další vývojové nástroje.

JDK obsahuje JRE (JVM a API), překládač, debbuger pro ladění programů, nástroj javadoc pro generování dokumentace, nástroj pro vytváření JAR archivů a další nástroje.

Z toho plyne, že pro spouštění Java programů potřebujeme mít nainstalované JRE, ale pokud budeme chtít Java programy I vytvářet a kompilovat, budeme potřebovat JDK.

Programovací jazyk Java definuje několik platforem. Mezi ně patří Java ME pro aplikace provozované na mobilních zařízeních. Dále je to Java SE pro aplikace běžící na stolních počítačí a Java EE používaná pro podnikové, rozsáhlé a webové Java aplikace [4].

4.1. Instalace Javy

Abychom mohli Javu využívat jako vývojáři, potřebujeme si stáhnout a nainstalovat JDK. JDK stáhneme z adresy http://java.sun.com a na operačním systému Windows nainstalujeme podle zvyklostí.

Instalaci JDK na ditribuci Linuxu Ubuntu nainstalujeme příkazem

```
sudo apt-get install sun-java6-jdk
```

4.2. Nastavení systémových proměnných

Po nainstalování musíme nastavit několik proměnných o instalaci Javy.

4.2.1. JAVA_HOME

Tato proměnná musí obsahovat kořenový adresář instalace Javy.

Nastavení Windows

```
set JAVA_HOME=c:\Program Files\Java\jdk1.6.0_21
```

Nastavení Linux

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

4.2.2. PATH

Tato proměnná musí odkazovat na adresáře vývojových nástrojů.

Nastavení Windows

```
set PATH=%PATH%;c:\Program Files\Java\jdk1.6.0_21\bin
```

Nastavení Linux

```
export PATH=$PATH:/usr/lib/jvm/java-6-sun bin
```

4.2.3. CLASSPATH

Tato proměnná není povinná, ale je vhodné ji nastavit. Měla by obsahovat cesty ke spustitelným souborům, uživatelským knihovnám jako JAR soubory a podobně. Nastavení se provede obdobně jako proměnná `JAVA_HOME` a `PATH`.

4.3. Kompilace a spuštění programů

V této kapitole si popíšeme základní příkazy pro činnosti spojené s kompilací, spouštění, archivací java programů apod. Tyto příkazy hojně využijeme při vývoji Java komponent.

Mějme soubor `hello.java`, který uložíme do adresáře `d:\work\packages\hello_package`.

```
package hello_package;
public class hello {
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
```

V příkazové řádce se přesuneme do adresáře `d:\work\packages a` soubor `hello.java` zkompilujeme příkazem

```
javac hello_package\hello.java
```

Následně ho můžeme spustit příkazem

```
java hello_package.hello
```

Chceme-li vytvořit jar archiv, pak zapíšeme

```
jar cf hello.jar hello_package\hello.class
```

Pro zobrazení obsahu jar archivu zadáme

```
jar tf hello.jar
```

Jar archiv můžeme rozbalit příkazem

```
jar xf hello.jar
```

Jar archiv můžeme také spustit příkazem

```
java –jar hello.jar
```

5. Java komponenta

V této kapitole si ukážeme, jak z jednoduché matlab aplikace vytvoříme Java komponentu, která bude v podobě `.jar` souboru, jenž pak lze importovat do jiné Java aplikace a využívat její třídy a metody.

Jedná se o jednoduchou komponentu, která bere jako vstupní argument pole čísel a vrátí jejich součet.

Vytvoříme Java komponentu, která bude obsahovat třídu `Sum` s metodou `sumarum()`. Tato Java metoda zapouzdří Matlab funkci `sumarum`. Matlab funkce přijme libovolný počet hodnot a vrátí jejich součet.

Tvorba Java komponenty se skládá ze dvou částí. První část vyplňuje Matlab programátor, který píše Matlabovský kód, druhou část tvoří Java programátor, který píše Javovský kód využívající Java komponentu.

Zdrojové kódu této aplikace jsou v příloze na CD v adresáři `aplikace/Sumarum`.

5.1. Úkoly Matlab programátora

- Napsání Matlab kódu
- Otestování Matlab kódu, zda pracuje správně
- Vytvoření Java balíčku (zapozdření Matlab kódu do Java tříd) – spuštění funkce ***Build the project*** v nástroji ***deploytool***.
- Spuštění ***Packaging Tool***, který spojí Java komponentu s dalšími přidanými soubory
- Předání výstupního balíčku z Packaging Tool (adresář `distrib`) Java programátorovi.

5.1.1. Matlab funkce

Vytvoříme si pracovní adresář `D:\work\project_sum\matlab`, do kterého umístíme m-file s funkcí `sumarum`. Soubor `sumarum.m` obsahuje následující kód.

```
function soucet = sumarum(data)

soucet = sum(data);
```

Ten převezme vstupní pole a pomocí funkce `sum` vypočte součet všech hodnot v poli a výsledek vloží do návratové proměnné `soucet`.

5.1.2. Testování matlab souboru

V matlabu přejdeme do adresáře `D:\work\project_sum\matlab` a zapíšeme příkaz:

```
sumarum([1.5, 1.4, 0.3])
```

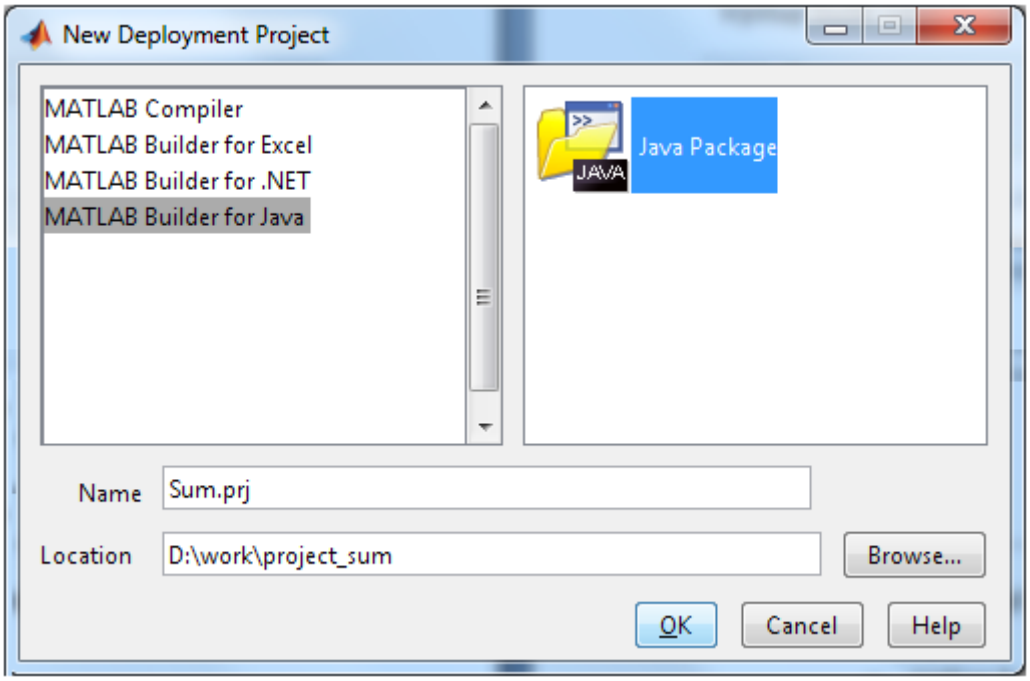
Výstup bude následující:

```
ans =

    3.2000
```

5.1.3. Vytvoření Java komponenty

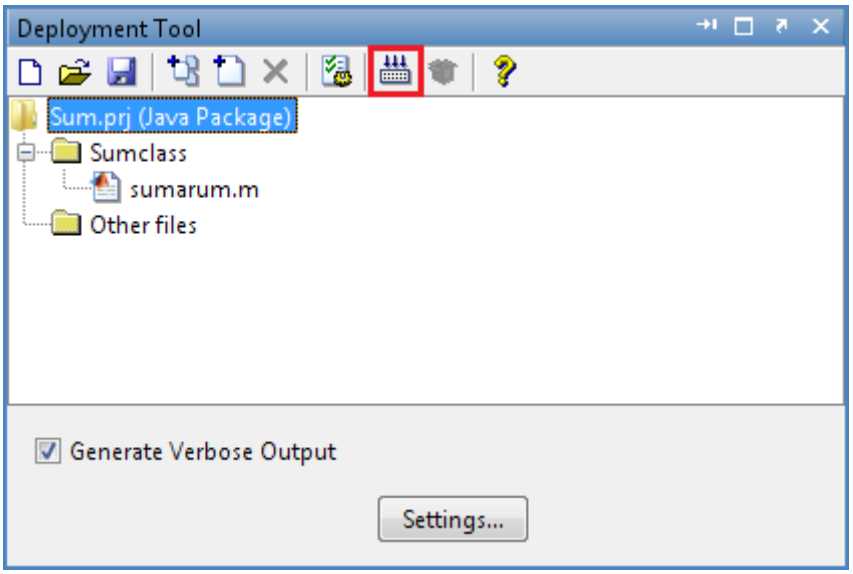
Nástroj pro vytvoření komponenty spustíme zadáním příkazu `deploytool` do příkazového řadku Matlabu. Následně vytvoříme nový projekt Java Builder JA podle následujícího obrázku.



Obr. 3: Vytvoření nového projektu Matlab BuilderJA

Po stisknutí na tlačíko OK se otevře nové okno se stromovou strukturou komponenty. Adresář ***Sumclass*** představuje třídu, do které vložíme Matlabovský soubor `sumarum.m` s matlab funkcí, která bude ve třídě dostupná jako metoda `sumarum()`. Název třídy můžeme upravit podle vlastního uvážení. Pokud soubor `sumarum.m` využívá funkce nějakého jiného m-filu, není nutné jej přidávat ručně.

O závislosti se stará Matlab Compiler a soubor do projektu přidá automaticky. Jméno projektu udává název JAR balíku, který se vytvoří. V tomto případě bude jméno Java balíku `Sum.jar`. Komponentu sestavíme stisknutím tlačítka ***Build the project***.



Obr. 4: Sestavení Java komponenty

5.1.4. Zabalení Java komponenty

Proces svázání Java komponenty s dalšími soubory do JAR souboru, který je distribuován koncovým uživatelům se nazývá balení. Balení můžeme provést nástrojem ***deploytool*** anebo zkopírováním obsahu složky `distrib` a MCR instalátou do složky dle našeho výběru. V případě využití nástroje deploytool pro zabalení komponenty provedeme následující kroky.

- V nastavení na kartě ***Packaging*** přidáme ***MATLAB Compiler Runtime*** zatrhnutím možnosti ***Include MATLAB Component Runtime (MCR)*** . Přitom máme dvě možnosti:
  - vložit MCR do balíčku – fyzicky zkopíruje instalační MCR do balíčku, který vytváříme,
  - vložit odkaz na MCR – nekopíruje MCR instalátor do balíčku, ale využije instalátor ze sítě. Tzn., že řekneme, kde v síti se instalátor nachází a až bude potřeba, tak jej použijeme. Tuto volbu využijeme, pokud máme hodně koncových uživatelů.
- Dále přiložíme soubory užitečné pro koncového uživatele jako `README.TXT` kliknutím na ***Add file/directory***.
- V `deploytool` provedeme zabalení kliknutím na tlačítko zabalení  .

V OS Windows je balíček samostatně spustitelný. Na jiných platformách se vytvoří ZIP soubor. Zkontrolujeme že v adresáři `distrib` jsou soubory, které jsme specifikovali.

Vytvořený balíček obsahuje následující části:

- sestavená komponenta,
- MCR instalátor, pokud byla tato volba použita v průběhu sestavování,
- dokumentace generovaná javadocem.

5.2. Úkoly JAVA programátora

Následuje práce pro Java programátora:

- přijme Java komponentu od Matlab programátora,
- otestuje funkčnost Java kódu, zda funguje stejně jako Matlab kód,
- nainstaluje MCR,
- do stávající Java aplikace importujeme třídy vygenerované matlab Builder JA,
- použije vestavěné třídy metod,
- řeší problémy s konverzí datových typů,
- zkontroluje, že výsledná aplikace funguje na koncovém zařízení tak jak má.

5.2.1. Instalace MCR

MCR je výkonný stroj, který se skládá ze sdílených knihoven Matlabu. Umožňuje spustit Matlab soubory a funkce na počítači, na kterém není Matlab nainstalovaný.

Instalace probíhá následně:

- instalace MCR,
- instalace komponent umístěných ve složce, ze které je instalace spuštěna,
- kopíruje MWArray do GAC (Global Assembly Cache) jako součást instalace MCR.

Instalace MCR vyžaduje přístup k systémovým registrům. Verze MCR musí být kompatibilní s verzí Matlab kompilátoru, který sestavil komponentu.

MCR nainstalujeme pomocí souboru `MCRInstaller.exe`, v případě, že MCR byl přidán při balení komponenty. V opačném případě zadáme příkaz

```
mcrinstaller
```

pro zobrazení míst, kde si můžeme instalátor stáhnout.

Pokud instalujeme MCR na platformu jinou než Windows, musíme nainstalovat MCR a nastavit systémové cesty podle následující dokumentace [5].

5.2.2. Testování java komponenty v Java aplikaci

Pro ověření funkčnosti Java komponenty vytvoříme program `SumDesktopClass.java`, který používá komponentu `Sum.jar`. Soubor `SumDesktopClass.java` uložíme do adresáře `D:\work\project_sum\classes\SumDesktop`.

Pokud voláme metodu Java komponenty z Java třídy, pak vstupní parametry předávané funkci musí být ve tvaru Matlabovských interních polí. V tomto případě můžeme parametry:

- manuálně konvertovat pomocí balíčku `com.mathworks.toolbox.javabuilder`,
- nebo předat jako datové typy jazyka Java. Ty budou automaticky převedeny.

Více informací o přetypování datových typů je v kapitole 9 s názvem MWArray. Obsah programu s komentářem je v následujícím výpisu.

```
package SumDesktop;

// import knihovny javabuilder.jar, která je potřebná pro komunikaci
// s Java komponentou.
import com.mathworks.toolbox.javabuilder.*;

// import Java komponenty
import Sum.*;

public class SumDesktopClass {

   public static void main(String[] args) throws MWException {
      // deklarace proměnné komponenty
      Sumclass sum = null;
      // deklarace proměnné pro uložení výsledku
      Object[] max = null;
      // testovací data získáme z příkazové řádky
      double[] data = new double[args.length];
      for(int i = 0; i < args.length; i++) {
         data[i] = Double.valueOf(args[i]);
      }
      try{
         // vytvoření instance Java komponenty
         sum = new Sumclass();
         // zavolání funkce komponenty, první argument představuje
         // očekávaný počet výstupních parametrů, druhý argument je
         // pole hodnot, komponenta vrátí pole typu Object, které
         // můžeme následně přetypovat např. na datový typ double
         max = sum.sumarum(1, data);
         System.out.println(max[0]);
      } catch (Exception e) {
         // V případě vyhození a zachycení vyjímky se vypíše hlášení
         // o chybě
         System.out.println("Exception: " + e.toString());
      } finally {
         // v této fázi se ruší objekty MWArray a objekty komponenty
         if (sum != null)
            sum.dispose();
      }
   }
}
```

Kompilace třídy pracující s Java komponentou vyžaduje znát cestu k souboům `javabuilder.jar` a `Sum.jar`. Tyto archivy uložíme do adresáře `D:\work\project_sum\lib`. Pro spuštění kompilace Java třídy přejdeme v příkazovém řádku do adresáře `D:\work\project_sum` a zapíšeme následující příkaz

Windows

```
%JAVA_HOME%\bin\javac -classpath
     .\lib\javabuilder.jar;
     .\lib\Sum.jar
     .\classes\SumDesktop\SumDesktopClass.java
```

Unix

```
$JAVA_HOME/bin/javac -classpath
     ./lib/javabuilder.jar:
     ./lib/Sum.jar
     ./classes/SumDesktop/SumDesktopClass.java
```

Příkazy zřetězíme znakem '`;`' na Windows a '`:`' na Unixu.

Popis komponent příkazů:

- `%JAVA_HOME%\bin\javac` – zavolání překladače,
- `-classpath` – tento přepínač umožňuje přístup k balíčkům a dalším souborům, které jsou potřebné ke kompilaci komponenty,
- `.\lib\javabuilder.jar` – umístění Matlab Builder JA balíčku,
- `.\lib\Sum.jar` – umístění `Sum.jar` balíčku ,
- `.\classes\SumDesktop\SumDesktopClass.java` – umístění zdrojového souboru `SumDesktopClass.java`.

Následně spustíme aplikaci

Windows

```
%JAVA_HOME\bin\java -classpath
     .\classes;
     .\lib\javabuilder.jar;
     .\lib\Sum.jar
     SumDesktop.SumDesktopClass 1.5, 1.4, 0.3
```

Linux

```
$JAVA_HOME/bin/java -classpath
     ./classes:
     ./lib/javabuilder.jar:
     ./lib/Sum.jar
     SumDesktop.SumDesktopClass 1.5, 1.4, 0.3
```

Popis komponent příkazu

- `%JAVA_HOME%\bin\java` – explicitní zavolání Java Run Time z Matlab JRE,
- `-classpath` – toto umožňuje přístup k balíčkům a dalším souborům, které jsou potřebné pro spuštění programu,
- `.\classes;` - umístění balíčku `SumDesktop`,
- `.\lib\javabuilder.jar;` - umístění Matlab Builder JA balíčku,
- `.\lib\Sum.jar` – umístění `Sum.jar` balíčku vytvořeného pomocí ***deploytool***,
- `SumDesktop.SumDesktopClass 1.5, 1.4, 0.3` – spuštění třídy a předání hodnot.

Po spuštění zkontrolujeme výpis, zda souhlasí s výstupem Matlabu.

```
3.200
```

Celá tato kapitola byla napsána podle dokumentace firmy mathworks na stránkách Matlab Builder JA [6].

6. Aplikační server

Pro nasazení Java komponenty na web musíme mít nainstalovaný webový server Apache. Samotný webový server Apache ale nestačí pro spouštění servletů. K tomuto účelu je potřeba nainstalovat a zprovoznit také aplikační server, který umožňuje spolupráci se servlety.

Servlet kontejner je speciální komponenta aplikačních serverů, která podporuje spouštění servletů. Jednotlivé servlety se registrují u servlet kontejneru, podávají mu informace o funkci, kterou poskytují a URL nebo jiný identifikátor, který bude používán pro jeho identifikaci. Servlet kontejner inicializuje servlet a předává mu parametry z požadavku. Servlet kontejnery jsou označovány také jako web kontejnery nebo web enginy.

Je mnoho implementací takovýchto aplikačních serverů jako například Tomcat, Manager, JBoss, Jetty a další. Pro nasazení Matlab serveru na naší katedře jsem zvolil Tomcat pro svou jednoduchost a rychlost.

6.1. Tomcat

Apache Tomcat (dříve Jakarta Tomcat) je open source web kontejner používaný podle oficiální referenční implementace pro Java Servlet a Java Server Page technologie (JSP).

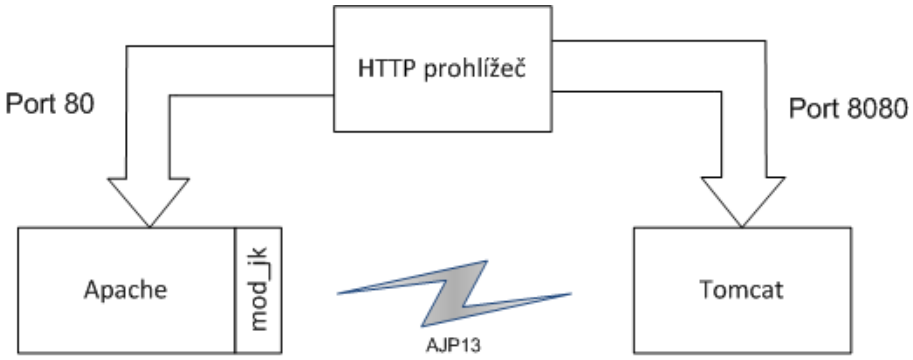
Dalším důvodem proč jsem zvolil Tomcat je, že Tomcat umí pracovat jako samostatný webový server, ale také jako plug-in do webového serveru Apache. Apache totiž mnohem rychleji obsluhuje požadavky na statické stránky než to dokáže Tomcat. Proto se používá Apache pro obsluhu statického obsahu. Požadavky směřující na Java aplikace směruje Apache na Tomcat.

V této práci je Tomcat využit jako plug-in webového serveru Apache. Pokud si webový prohlížeč zažádá o klasickou HTML nebo PHP stránku, pak požadavek obslouží samotný Apache. Pokud bude webový prohlížeč požadovat servlet, požadavek přijme webový server Apache a následně jej přepošle na Tomcat, který tento požadavek zpracuje.

Pro spojení Tomcatu s Apachem se používá konektor ***mod_jk***. Komunikace mezi Apachem a Tomcatem probíhá na základě protokolu ***AJP13***. Jedná se o optimalizovanou verzi http protokolu.

Apache/Tomcat budu označovat použití Tomcatu jako rozšíření webového serveru Apache. Tomcat/Apache se skládá ze dvou nezávislých částí. První je modul Apache napsaný v C, který umožňuje komunikaci Apache s Tomcatem. Jedná se o již výše zmiňovaný

konektor, jenž je na obrázku označen jako modul mod_jk. Druhou část tvoří kód v Javě, který implementuje Java servlet API v JVM. Tato část je samotný Tomcat. Obě dvě části spolu komunikují pomocí protokolu AJP13.



Obr. 5: Schéma komunikace Apache/Tomcat

Tomcat může být použit jako aplikační server pro méně náročné Java EE webové aplikace. Tomcat je velice oblíbený z hlediska podpory následujících komponent:

- WAR soubory,
- JDBC zdroje dat,
- JSP stránky,
- virtuální hosty,
- clustrování aj.

Tomcat je velice výkonný a konkurence schopný aplikační server pro jiné placené aplikační servery. Tomcat ve verzi 6 podporuje JSP 2.1 a Servlety 2.5 a další nové funkce. Stále ještě neumí tolik, kolik nabízí Java EE. Mezi funkce, které Tomcat stále ještě neumí je například:

- distribuované transakce,
- EJB,
- JMS apod.

Pro potřeby funkcí, které Tomcat neumí, je potřeba využít aplikačních serverů jako Jboss, WebSphere a jiné. Rozdíl mezi enterprise aplikacemi (Java EE) běžících na aplikačních serverch a webovými aplikacemi (Java SE) je rozepsán v příloze na CD `priloha/enterprise-webove-aplikace.pdf`.

6.2. Protokol ajp13

Webový server komunikuje se servletovým kontejnerem přes TCP spojení. K této komunikaci se používá protokol AJP13 vycházející z protokolu HTTP. Pro snížení zátěže lze zachovat

trvalé spojení TCP se servletem a využít jej pro více cyklů požadavek/odpověď jednoho sezení. Jakmile je TCP spojení přiřazeno určitému požadavku, pak toto spojení nelze využít pro obsloužení jiného požadavku jiného klienta. Jinými slovy, nelze využít multiplexovaný přístup po jednom spojení.

6.3. Tomcat workers

V některých případech je potřeba více instancí aplikačního serveru Tomcat. Instance Tomcatu se nazývá ***worker***. V další části budu používat výraz worker. Může se jednat například o situaci, kde ve vývojovém týmu pracuje více lidí, kteří implementují Java EE aplikace a potřebují každý svůj aplikační server Tomcat. Nebo pokud provozujeme webový server s virtuálními hosty, pak pro každého virtuálního hosta chceme přidělit jeden Tomcat z důvodu oddělení prostorů jednotlivých webů mezi různými společnostmi. Jako další možnost použití více workerů Tomcatu je potřeba rozložit zátěž na více počítačů. Pak jednotlivé workery budou na různých strojích.

6.4. Instalace Tomcat

Windows

Na systémech Windows si stáhneme ze stránek

`http://www.apache.org/dist/tomcat/tomcat-5/v5.5.33/bin/`

instalační soubor `apache-tomcat-5.5.33.exe` a nainstalujeme. V průběhu instalace budeme dotázání pro uživatelské jméno a heslo pro přístup k webovému rozhraní Tomcatu.

Linux

Na operačním systému Linux je vhodné instalovat Tomcat z binární distribuce. Binární distribuce

v sobě totiž už obsahuje balíčky tříd Javy, které jsou potřebné pro běh servletů na serveru. Při použití instalace ze zdrojového kódu nemůžeme použít standardní nástroj make ale sestavovací nástroj Ant. Binární aktuální verzi stáhneme příkazem `wget`.

```
wget http://apache.miloslavbrada.cz/tomcat/tomcat-5/v5.5.33/bin/apache-
tomcat-5.5.33.tar.gz
```

Následně archiv rozbalíme a přesuneme do připraveného adresáře /usr/local/tomcat

```
tar xvzf apache-tomcat-5.5.33.tar.gz
sudo mv apache-tomcat-5.5.33 /usr/local/tomcat
```

Nyní si vytvoříme skript pro spouštění, vypínání a restartování aplikačního serveru Tomcat. Obsah skriptu, který následně uložíme do souboru `/etc/init.d/tomcat` je v příloze na CD `priloha/zdrojove_kody/tomcat`.

Následně změníme práva souboru a vytvoříme symbolické odkazy.

```
sudo chmod 755 /etc/init.d/tomcat
sudo ln -s /etc/init.d/tomcat /etc/rc1.d/K99tomcat
sudo ln -s /etc/init.d/tomcat /etc/rc2.d/S99tomcat
```

Tomcat spustíme, vypneme nebo restartujeme následujícími příkazy.

```
/etc/init.d/tomcat start
/etc/init.d/tomcat stop
/etc/init.d/tomcat restart
```

6.4.1. Ověření funkčnosti

Před samotným ověřením funkčnosti Tomcatu musíme zkontrolovat proměnné prostředí:

- `JAVA_HOME` – umístění Javy,
- `JRE_HOME` – umístění běhového prostředí Javy (`JAVA_HOME\jre`),
- `PATH` umístění nástrojů Javy (`JAVA_HOME\bin`),
- `CATALINA_HOME` umístění Tomcatu.

V této chvíli zadáme do webového prohlížeče adresu ***localhost:8080*** a zobrazí se nám stránka webového rozhraní Tomcatu.

6.5. Tomcat jako plug-in Apache

Aby Apache odesílal požadavky pro zpracování Tomcatu, musíme mít takový modul, který má přístup k Tomcatu na jednom z jeho konektorů. Takových konektorů existuje celá řada, např. Coyote http/1.1, JK, Webapp a další. Jak jsem již uvedl, budu používat právě konektor JK, protože je doporučovaný na stránkách vývojářů Tomcatu.

6.5.1. Instalace mod_jk

Linux

Modul mod_jk nainstalujeme následujícím příkazem.

```
sudo apt-get install libapache2-mod-jk
```

Windows

Konektor JK si stáhneme v podobě binarní verzi z adresy

`http://www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/`.

Stažený soubor aktuální verze s názvem např. `mod_jk-1.2.31-httpd-2.2.3.so` přejmenujeme na `mod_jk.so` a nakopírujeme do adresáře modulů v Apachi.

6.5.2. Konfigurace Tomcat - Windows

soubor worker.properties

`workers.properties` je povinný konfigurační soubor, který využívá webový server a je stejný pro všechny implementace JK.

Zde se konfigurují jednotlivé workery. V directivě `worker.list` definujeme seznam workerů oddělených mezerami. Pro každý worker nastavíme typ používaného protokolu, jméno počítače, na kterém běží a port na kterém poslouchá.

Zde je minimální konfigurace využívající ajp13 pro spojení webového serveru Apache s Tomcatem jako plug-in.

```
# Definování jedné instance Tomcatu s názvem worker1
worker.list=worker1
# Nastavení hodnot pro worker1 využívající ajp13 pro spojení
# Apache s Tomcatem jako plug-in
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
```

Používané atributy pro workery

Atributů existuje celá řada [8], ty nejpoužívanější jsou vypsány níže.

- `type` – typ protokolu (ajp13),
- `host` – název nebo IP adresa hostitele, na kterém je umístěn Tomcat,
- `port` – číslo portu, na kterém naslouchá worker. Defaultně pro ajp13 to je 8009,
- `socket_timeout` – čas v sekundách, po kterou bude modul JK čekat na odpověď vzdáleného hostitele, pokud v této době nezíská odpověď, vygeneruje chybu a pošle znovu žádost,
- `ping_mode` – nastavení jakým způsobem se bude testovat, zda je vzdálený hostitel dostupný a pracuje. Lze použít následující módy. ***C*** – ping se provede při každém spojení ke vzdálenému hostiteli. ***P*** – ping se provede před každým odesláním požadavku. ***I*** – ping se provádí ve chvíli, kdy je vzdálený hostitel nečinný delší dobu než je uvedeno v `connection_ping_interval`.
- `ping_timeout` – doba v milisekundách, po kterou čeká odpověď na ping,
- `connection_ping_interval` – čas v sekundách, který udává jak dlouho musí být vzdálený hostitel neaktivní, aby se na něj poslal ping v případě nastavení `ping_mode=I`,

Soubor `httpd.conf`

Na konec souboru konfiguračního souboru Apache anebo do externího souboru, který importujeme do souboru `httpd.conf` napíšeme následující konfiguraci.

Nahrání modulu `mod_jk.so`.

```
LoadModule    jk_module  modules/mod_jk.so
```

Včlenění konfiguračního souboru `worker.properties`.

```
JkWorkersFile d:\matlabserver\xampp\apache\conf\workers.properties
```

Soubor do kterého se budou zapisovat logy.

```
JkLogFile     d:\matlabserver\xampp\apache\log\mod_jk.log
```

Nastavení úrovně logování [debug/error/info]

```
JkLogLevel    info
```

Nastavení formátu času v logovacím souboru

```
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
```

Určí ty požadavky, které se mají přeposlat worker se jménem `worker1` (ajp13). V tomto případě to budou ty požadavky, které budou začínat na `servlet-examples`.

```
JkMount  /servlet-examples/* worker1
```

6.5.3. Konfigurace Tomcat – Linux

Konfiguraci na operačním systému Linux zahájíme příkazem

```
dpkg -L libapache2-mod-jk
```

kterým zjistíme soubory, jež patří k nainstalovanému balíčku modulu `mod_jk`. Nejdůležitější je samotný soubor s modulem `/usr/lib/apache2/modules/mod_jk.so`, který se umístil do adresáře modulů Apache. Dále je to konfigurační soubor `/etc/libapache2-mod-jk/workers.properties`, ve kterém je umístěna konfigurace workerů. Buď tento soubor překopírujeme do adresáře `/etc/apache2/conf` a upravíme obdobně jako u Windows distribuce anebo vytvoříme nový soubor `/etc/apache2/conf/workers.properties` a vložíme do něj následující konfiguraci.

```
workers.tomcat_home=/usr/share/tomcat
workers.java_home=/usr/lib/jvm/java-6-sun
worker.list=worker1
worker.worker1.port=8009
worker.worker1.host=localhost
worker.worker1.type=ajp13
worker.worker1.lbfactor=1
```

Dále ještě vytvoříme konfigurační soubor `/etc/apache2/conf/tomcat.conf` do kterého vložíme následující konfiguraci.

```
JkWorkersFile    /etc/apache2/conf/worker.properties
JkLogFile        /var/log/apache2/mod_jk.log
JkLogLevel       info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
JkMount          /servlets-examples/* worker1
```

Ještě musíme začlenit tento soubor do konfigurace Apache přidáním následujícího řádku do souboru `/etc/apache2/httpd.conf`.

```
Include /etc/apache2/conf/tomcat.conf
```

Nakonec stačí jen povolit modul `mod_jk` a přinutit Apache aby znovu zavedl moduly.

```
sudo a2enmod jk
sudo /etc/init.d/apache2 force-reload
```

V této fázi je Tomcat nakonfigurován jako plug-in do Webového serveru Apache. Ověříme si to zadáním následují adresy do webového prohlížeče

```
localhost/servlets-examples
```

Tato kapitola byla vypracována podle literatury [9], [10], [11] a další informace o konfiguraci Tomcatu a konektoru `mod_jk` najdete na webu The Apache Software Foundation [7], [8].

7. Applety

Applet je zkompilovaný javovský kód, který se vloží do webové stránky, kde se spustí. Podmínkou spuštění appletu na webové stránce je, aby třída appletu byla potomkem třídy `java.applet.Applet`. Applety mají bezpečnostní omezení oproti dekstopovým Java aplikacím. Jedná se o následující omezení:

- applet nemůže nahrávat knihovny ani definovat nativní metody,
- applet nemůže navazovat síťové spojení s jiným než domovským serverem,
- applet nemůže zapisovat do souborů na straně klienta,
- applet nemůže spouštět programy na domovském serveru,
- applet nemůže číst některé systémové proměnné.

Kromě těchto omezení mají applety implementovány některé rozšiřující funkce:

- applet může přehrávat zvuky,
- applet může požádat prohlížeč o zobrazení libovolné WWW stránky.

7.1. Struktura appletu

Applet je javovská třída umístěná do webové stránky, která je potomkem třídy `java.applet.Applet`. Tato třída poskytuje rozhraní pro komunikaci mezi prohlížečem a appletem.

Prohlížeč volá metody v závislosti na životní fázi appletu:

- `public void init()` – při inicializaci,
- `public void start()` – při spuštění,
- `public void paint(java.awt.Graphics g)` – při překreslování,
- `public void stop()` – při zastavení,
- `public void destroy()` – při ukončení.

Tyto metody jsou ve třídě `java.applet.Applet` definovány jako prázdné a pro použtí appletů je potřeba tyto metody překrýt.

7.2. Ladění appletů

Applet umístěný do webové stránky se zobrazí při načtení webové stránky. V případě změny v appletu se musí restartovat prohlížeč. Pro ladění appletů je výhodné využít `appletviewer`, jež je součástí instalace javy. Applet umístěný ve webové stránce spustíme příkazem

```
appletviewer applet.html
```

7.3. Umístění appletu do webové stránky

Následuje kód zobrazující v HTML stránce applet `helloApplet.class`.

```
<applet   code = "helloApplet.class"
          width = "300"
          height = "300"
          name = "helloApplet"
          codebase= "applets/"
          archive = "JARsoubor1.jar, … JARsouborN.jar"
          alt = "alternativní text">
   <param name = "name1" value= "value1">
   <param name = "name2" value= "value2">
</applet>
```

Následuje popis jednotlivých atributů použitých v tagu applet:

- `code` – jméno hlavní třídy appletu,
- `width` – udává šířku boxu, do kterého se bude applet na WWW stránce zobrazovat,
- `height` - udává výšku boxu, do kterého se bude applet na WWW stránce zobrazovat,
- `name` – jméno appletu, které lze získat metodou `getApplet()`, slouží pro komunikaci appletů,
- `codebase` – url, udávající absolutní nebo relativní cestu, kde se budou hledat třídy appletu a JAR soubory, není-li použito, je hodnotou `codebase` adresář, odkud byl načten HTML soubor odkazující na applet,
- `archive` – jeden nebo více archivů, které se načtou při načtení stránky,
- `alt` – alternativní text, který se zobrazí v případě, kdy se rozezná značka applet, ale není podporovaný grafický režim.

7.4. Applet jako uživatelské rozhraní servletů

Servlety většinou provádějí nějakou akci, výpočet na základě vstupních hodnot. Pro zadání lze použít HTML formuláře. Tyto formuláře umožňují zadávat hodnoty do textových polí, můžeme využít zaškrtávací pole, seznamy, či odeslání souboru. Takto získaná data se odesílají servletu.

Applety umožňují vytvářet mnohem dokonalejší formuláře, než ty, které jsou vytvořené pomocí HTML. S applety máme větší možnosti ovlivnit rozměry, barvu či font řídících prvků GUI, poskytují více schopností jako posuvníky, kreslení čar, dialogová okna, apod. Umožňují sledování myší a klávesnice, přetahování ikon apod. Výsledky, kterých lze dosáhnout s HTML formuláři a applet formuláři je obrovský. Daň, kterou za to zaplatíme je v úsilí, které musíme vynaložit pro vytvoření takového appletu v Javě.

7.4.1. Techniky odesílání dat na server

U appletů existují tři různé přístupy pro odesílání dat k servletu a prezentace výsledných dat.

V případě prvního přístupu se data odesílají metodou GET a servlet vytvoří výslednou stránku, kterou pak prohlížeč zobrazí.

V případě druhého přístupu se také použije metoda GET, ale servlet nevytváří celou stránku, ale jen výsledky. Applet se pak postará o vytvoření stránky, do které umístí výsledky získané ze servletu. Tento přístup je rychlejší, neboť se nepřenáší od servletu celá stránka, ale jen výsledky.

Posledním přístupem je použití metody POST pro odeslání dat, kdy applet výsledky získané servletem sám zpracuje a zobrazí.

7.4.2. Komunikace metodou GET

Tuto techniku využívám ve svých aplikacích pro načtení grafiky vytvořené na serveru. Na servlet se pouze odešle požadavek, servlet vytvoří výstupní grafiku, kterou potom applet zobrazí. Kód na straně appletu při této komunikaci se servletem vypadá následně.

```
String params  = "&param=" + value;
String stringUrl = "http://domain/servlet" + params;
try {
   URL programURL = new URL(stringUrl);
   String target = "content_appl";
   getAppletContext().showDocument(programURL, target);
} catch(MalformedURLException mue) {
   // ošetření vyjímky
}
```

Nejprve se vytvoří object URL servletu, na který se budou odesílat požadavky. Metoda `getAppletContext()` získá odkaz na aktuální applet a metodou `showDocument()` se vytvoří požadavek na stažení požadovaného dokumentu, který je následně zobrazen v okně nebo rámu.

Podmínkou fungování je, aby byl servlet na stejném serveru, ze kterého byl applet zaveden.

7.4.3. Komunikace metodou POST

Při použití této techniky je také podmínkou, aby servlet byl na stejném serveru, ze kterého byl applet zaveden. A dále, že po appletu je požadováno, aby sám zobrazil výsledky. Výhodou je pak to, že servlet může být jednodušší, bez potřeby zabalit data do HTML stránky. K další výhodě patří to, že lze použít serializované datové proudy. Tzn., že je možné posílat na servlet

a číst z něj přímo javovské objekty. Tyto objekty se posílají jako binární data. Pro poslání dat na servlet a přečtení výsledků je zapotřebí následujících 13 kroků.

Vytvoření obejktu URL servletu

```
URL aktualniStranka = getCodeBase();
String protokol = aktualniStranka.getProtocol();
String host = aktualniStranka.getHost();
int port = aktualniStranka.getPort();
String urlSuffix = "servlets/NejakyServlet";
URL dataURL = new URL(protocol, host, port, urlSuffix)
```

Vytvoření objektu URLConnection

Tento objekt se použije pro získání vstupních a výstuních proudů, které se připojují k serveru.

```
URLConnection connection = dataURL.openConnection();
```

Data neukládat do mezipaměti

Dále řekneme prohlížeči, aby data neukládal do vyrovnávací paměti.

```
connection.setUseCaches(false);
```

Povolení odesílání dat

Následně musíme systému sdělit, aby nám povolil data odeslat, ne jen přečíst.

```
connection.setDoOutput(true);
```

Vytvoření ByteArrayOutputStream

Tento objekt slouží pro uložení dat, která budou serveru odeslána, do vyrovnávací paměti. Velikost není důležitá. V případě potřeby se velikost automaticky zvětší.

```
ByteArrayoutputStream byteStream = new ByteArrayOutputStream(512);
```

Připojení výstupního proudu k ByteArrayOutputStream

Pro odeslání normálních formulářových dat použijeme `PrintWriter`. Pro odeslání serializovaných datových struktur použijeme místo toho `ObjectOutputStream`.

```
PrintWriter out = new PrintWriter(byteStream, true);
```

Uložení výstupních dat do vyrovnávací paměti

Pro formulářová data použijeme `print`. Pro serializované datové struktury použijeme `writeObject`. Metoda `flush()` zajistí odeslání všech dat uložených do vyrovnávací paměti.

```
String hodnota1 = URLEncoder.encode(nejakaHodnota1);
String hodnota2 = URLEncoder.encode(nejakaHodnota2);
String data = "prom1=" + hodnota1 + "&prom2=" + hodnota2;
out.print(data);
out.flush();
```

Nastavení záhlaví Content-Length

Používáme-li metodu POST je nutno nastavit záhlaví `Content-Length`.

```
connection.setRequestProperty("Content-Length",
   String.valueOf(byteStream.size()));
```

Nastavení záhlaví Content-Type

Při odesílání formulářových dat je zapotřebí nastavit hodnotu na `application/x-www-form-urlencoded`.

```
connection.setRequestProperty("Content-Type",
   "application/x-www-form-urlencoded ");
```

Odeslání skutečných dat

```
byteStream.writeTo(connection.getOutputStream());
```

Otevření vstupního proudu

Pro textová nebo binární data použijeme `BufferedReader`, pro serializovaná data použijeme `ObjectInputStream`.

```
BufferedReader in = new BufferedReader(new InputStreamReader
   (connection.getInputStream()));
```

Čtení výsledků

Následuje přečtení dat odeslaných servletem.

```
String line;
while((line = in.readLine()) != null) {
   zpracuj(kline);
}
```

8. Servlety

Servlety jsou analogií k CGI skriptům. Servlety jsou napsány v Javě, které běží na straně serveru a vytvářejí dynamické HTML stránky, které se zobrazují ve webových prohlížečů klientů. Programátor servletů potřebuje knihovny, jež jsou součástí platform J2EE (Java 2 Enterprise Edition). J2EE obsahuje třídu `HttpServlet` a jediné co musí programátor udělat je, předefinovat její metody pro obsluhu jednotlivých typů požadavků.

Servlety se hodí, pokud se stránky dynamicky často mění, což je zcela častý případ. V těchto případech se používají JSP stránky (Java Server Page). Jsou to obyčejné HTML stránky, ve kterých jsou vloženy direktivy určující dynamický obsah. Webový server podporující servlety a JSP stránky nejprve JSP stránku přeloží, jejím překladem vznikne servlet, který se pak na webovém serveru spustí.

Servlet se jeví jako střední vrstva mezi požadavky od klientů a databází či aplikací na serveru http. Jejich úkolem je:

- přečíst data odeslaná uživatelem (z formuláře, z javovského appletu),
- vyhledat všechny další informace o požadavku – informace o prohlížeči, cockie, atd.,
- vytvořit výsledky, tento proces představuje komunikaci s databází, vyvolání další aplikace nebo přímý výpočet,
- vložení výsledků do HTML stránky,
- odeslání dokumentu zpět klientovi.

8.1. Výhoda servletů před CGI

Tradiční CGI spouští nový proces pro každý požadavek http. Je-li program krátký, pak administrativa spuštění může dominovat nad časem jeho provádění. Při použití servletů zůstává v provozu javovský virtuální stroj, který obsluhuje každý požadavek pomocí lehkého javovského vlákna, nikoli pomocí procesu zatěžující operační systém. Pokud bude N současných požadavků na stejný program CGI, systém zavede kód do paměti N-krát. Avšak při použití servletů se vytvoří N vláken, ale jen jedna kopie třídy servletu.

Další výhodou je, že několik servletů může sdílet stejná data, jako připojení k databázi apod. Dále obsahují robustné API pro práci s požadavky, odpověďmi, cookies, sledování sezení, zpracovávání formulářů apod.

Obrovskou výhodou je přenositelnost servletů mezi různými platformami serverů, které podporují servlety. To je zaručeno tím, že zkompilovaný program v Javě není orientovaný na

platformu, ale na JVM (Java virtual Machine). Tento kód je překládán do strojového kódu až při spuštění na dané platformě.

8.2. Spolupráce mezi servlety

Pro zpracování http požadavku lze využít jeden servlet anebo několik servletů, které si v průběhu zpracovávání předávají řízení. Toho se využívá u sofistikovaných webových aplikací, kde se jeden servlet postará o zpracování příchozích informací, následně vybere servlet generující HTML stránku a předá mu řízení. Tento servlet může volat další různé servlety pro generování různých částí obsahu jako hlavička, navigační lišta, obsah, patička apod.

O předávání řízení mezi servlety se stará třída `javax.servlet.RequestDispatcher`. Tato třída umožňuje předat řízení buď úplně pomocí metody `forward()`, nebo jen dočasně metodou `include()`.

8.3. Rozsah platnosti atributů

Servlety mohou ukládat informace do atributů. Dále pak spolupracující servlety si mohou předávat pomocí těchto atributů informace. Nastavení těchto atributů se provádí metodou `setAttribute()`. Využívá se několik oborů platnosti těchto atributů.

Úroveň	**Nastavení atributu**	**Existence**
aplikace	ServletContext.setAttribute()	v rámci celé aplikace, mezi všemi uživateli
session	HttpPSession.setAttribute()	ve všech voláních jednoho uživatele
HTTP požadavek	HttpServletContext.setAttribute()	jen po dobu trvání jednoho http požadavku
JSP stránka	JspContext.setAttribute()	jen v rámci jedné JSP stránky

Tab. 1: Rozsah platnosti atributů

8.4. Listenery

Pomocí listenerů můžeme nastavit webovým aplikacím to, že budou dostávat upozornění v případě, že dojde k určité události. Seznam událostí, které lze hlídat je v tabulce 1 v příloze na CD `priloha/tabulky.pdf`.

8.5. Instalace a nastavení

K provozu servletů potřebujeme webový server, který podporuje servlety. K dispozici jsou např. následující:

- Apache Tomcat,
- JavaServer Web Development Kit (JSWDK),
- Allaire JRun,
- Java Web Server od firmy Sun.

V mé práci využívám aplikační server Apache Tomcat, jak bylo popsáno v kapitole 6.

8.5.1. Identifikace tříd pro kompilátor Javy

Pokud máme nainstalovaný software, musíme sdělit kompilátoru `javac`, kde najde soubory tříd, servletů a JSP, když kompiluje servlety. Nutné soubory se většinou nachází v podadresáři `lib` instalačního adresáře serveru se servletovými třídami `servlet-api.jar` a dalšími. Nejsnadnější způsob jak informovat `javac` o umístění těchto souborů je pomocí `CLASSPATH`. Jedná se o proměnnou, která specifikuje, kde `javac` vyhledává třídy při kompilaci. Není-li specifikovaná, pak se `javac` podívá do aktuálního adresáře a standardních systémových knihoven. Proměnná `CLASSPATH` musí mít také nastaveno "." jako aktuální adresář.

Následují informace, jak nastavit `CLASSPATH` na různých OS.

Windows

```
set CLASSSPATH=.;dir\servlet-api.jar;$CLASSPATH$
```

Unix

```
setenv CLASSPATH .:dir/servlet-api.jar:$CLASSPATH$
```

8.6. Konfigurace serveru

8.6.1. Číslo portu

Číslo portu identifikuje port, na kterém bude náš server naslouchat. Pokud nepoužíváme žádný jiný webový server, pak nastavíme port na 80. Pokud ale nějaký server provozujeme, nastavíme nějaké jiné číslo portu, např. 8080. Nastavení se provádí v souboru `instal_dir/conf/server.xml`.

8.7. Základní struktura servletu

Nejdůležitější součástí servletu je soubor se samotným servletem `HelloServlet.class` a konfigurační soubor `web.xml`. Servlet na webu má tuto struktůru.

```
simple_servlet/WEB-INF/classes/HelloServlet.class
simple_servlet/WEB-INF/web.xml
simple_servlet/index.html
```

8.8. Kompilace servletu

Kompilace zdrojových kódů servletů se provádí následujícím příkazem v případě, že v proměnné `CLASSPATH` jsou uvedeny cesty, v nichž se nachází všechny potřebné knihovny pro kompilaci. V opačném případě je nutné přepínačem `-classpath` uvést umístění těchto knihoven, jako to bylo v kapitole 5.

```
javac ServletName.java
```

Soubory `.class` se nahrají do některého z těchto adresářů, podle verze Tomcatu, kde `install_dir` odpovídá obsahu proměnné `CATALINA_HOME`, tedy umístění instalace Tomcatu.

```
install_dir/webapps/WEB-INF/classes
install_dir/classes
install_dir/webapps/ROOT/WEB-INF/classes
```

Soubory `.jar` se uloží do adresáře

```
install_dir/lib
```

8.9. Servlet - HelloServlet

Nyní si vytvoříme servlet, který přijme požadavek a vytvoří výstupní html stránku s textem ***'Hello, World!'***. Zdrojové kódy jsou v příloze na CD v adresáři `aplikace/HelloServlet`.

Abychom získali servlet, musíme odvodit třídu od `HttpServlet` a předefinovat `doGet` nebo `doPost` v závislosti na tom, zda jsou data odesílána pomocí metody Get nebo Post. Jestliže chceme, aby servlet manipuloval s požadavky GET tak s POST a aby pro každý požadavek provedl stejnou činnost, tak jednoduše použijeme `doGet` a v metodě `doPost` zavoláme funkci `doGet`.

Obě tyto metody mají dva argumenty:

- `HttpServletRequest`,
- `HttpServletResponse`.

`HttpServletRequest` má dvě metody, které umožňují zjistit příchozí informace jako formulářová data, záhlaví požadavku http a jméno hostitelského počítače.

`HttpServletResponse` umožňuje specifikovat výchozí informace jako stavové kódy http (200, 404, …), záhlaví odezvy (Content-Type, Set-Cookie, …). Dále obsahuje třídu `PrintWriter` pro odesílání obsahu dokumentu. Tato třída obsahuje metodu `println`, pro naplnění dokumentu obsahem.

Protože `doGet` a `doPost` mohou vyvolat dvě vyjímky, je potřeba je zahrnout do deklarace. Nakonec je potřeba importovat balíky `java.io` (pro `PrintWriter`), `javax.servlet` (pro `HttpServlet`) a `javax.servlet.http` (pro `HttpServletRequest` a `HttpServletResponse`).

V následujícím výpise je kód, který vytvoří HTML stránku s obsahem ***Hello World***, který odešle nazpět prohlížeči.

Pomocí metody `setContentType` nastavíme typ odesílaných dat na html. Javovský kód servletu je na následujícím výpisu.

```
// import balíčku pro vstupně výstupní operace
import java.io.*;

// import balíčku pro obsluhu požadavků na servlet a vytváření odpovědí
import javax.servlet.*;
import javax.servlet.http.*;

// Třída servletu musí dědit ze třídy HttpServlet
public class HelloServlet extends HttpServlet {
   // překrytí metody doGet pro zpracování požadavku
   // předávají se 2 parametry
   // objekt HttpServletRequest představující požadavek
   // objekt HttpServletResponse představující odpověd
   public void doGet(HttpServletRequest request,
             HttpServletResponse response)
         throws ServletException, IOException {
      // nastavení hůlavičky - typ dokumentu odpovědi
      response.setContentType("text/html");
      // získání objektu pro zapisování do výstupu
      PrintWriter out = response.getWriter();
      // vytvoření výstupní stránky, která se odešle zpět klientovi
      String docType =
           "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
           "Transitional//EN\">\n";
      out.println(docType +
           "<HTML>\n" +
           "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
           "<BODY BGCOLOR=\"#FDF5E6\">\n" +
           "<H1>Hello World</H1>\n" +
           "</BODY></HTML>");
   }
}
```

Tento soubor přeložíme do bytekódu následujícím příkazem

```
javac HelloWorld.java
```

Výsledný soubor `HelloWorld.class` uložíme do adresáře

```
simple_servlet/WEB-INF/classes
```

Nyní vytvoříme konfigurační soubor `web.xml`. V tomto souboru se definuje název servletu `<servlet-name>`. Tento název se mapuje na class soubor `<servlet-class>`, který se má spustit a dále se název servletu mapuje na URL `<url-pattern>`, přes níž je servlet dostupný. Soubor `web.xml` vypadá následovně.

```
<?xml version="1.0" encoding="UTF-8"?>
  <web-app version="2.4"
     xmlns="http://java.sun.com/xml/ns/j2ee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <servlet>
     <servlet-name>simple_servlet</servlet-name>
     <servlet-class>HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
     <servlet-name>simple_servlet</servlet-name>
     <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

V tomto souboru je nadefinováno:

- `<servlet-name>` - jméno servletu – ***simple_servlet***,
- `<servlet-class>` - jméno třídy – ***HelloWorld***,
- `<url-pattern>` - url, na niž je servlet dostupný – ***hello***.

Servlet můžeme spustit prostřednictvím souboru `index.html`, jež obsahuje

```
<a href="hello">Vyzkouset</a>
```

Servlet můžeme spustit přímo zadáním do prohlížeče

```
localhost:8080/simple_servlet/hello
```

8.10. Objekt HttpServletRequest

Tento objekt obsahuje metody, pro zjištění informací z požadavku jako `getServerName()`, `getProtocol()` a další. Celý seznam metod je v tabulce 2 v příloze na CD

priloha/tabulky.pdf. Zadané parametry ve formuláři lze v servletu získat následující metodou getParameter("param").

8.11. Životní cyklus servletu

Při spuštění servletu se vytvoří jedna instance servletu a každý uživatelský požadavek vytvoří nové vlákno, které spustí metodu doGet nebo doPost.

Při spuštění servletu se spustí metoda init. Poté každý nový požadavek bude mít za následek vytvoření nového vlákna. Při vytvoření nového vlákna se spustí metoda service již vytvořené instance. Metoda service následně vyvolá metodu doGet nebo doPost nebo nějakou jinou v závislosti na typu požadavku.

8.11.1. Metoda init

Používá se pro inicializaci proměnných jednotlivých instancí. Existují 2 verze této metody. Jedna bez parametrů.

```
public void init() throws ServletException {
   // inicializační kód
}
```

Druhá verze se používá, když musí servlet číst specifické hodnoty nastavení serveru ještě dříve, než dokončí inicializaci. Může se například jednat o nastavení připojení k databázi apod.

```
public void init(ServletConfig config) throws ServletException {
   super.init(config)
   // inicializační kód
}
```

ServletConfig má metodu getInitParametr, pomocí níž lze získat inicializační parametry. Tyto parametry se ukládají do souboru web.xml.

Vyvolání metody init předka je velmi důležité. Objekt ServletConfig se totiž používá někde jinde a metoda init předka jej zaregistruje tam, kde jí může servlet později najít.

8.11.2. Metoda service

Tato metoda je vyvolána pokaždé, když uživatel odešle požadavek na servlet. Tato metoda poté vyvolá funkci doGet, doPost, doPut apod. v závislosti na typu požadavku.

8.11.3. Rozhraní SingleThreadModel

V případě že je vyvoláno více požadavků na jeden servlet, pak existuje více vláken daného servletu, které pracují s instančními proměnnými servletu. To by mohlo způsobit, že by vlákna přistupovali k těmto datům současně, což je nežádoucí stav. Rozhraní `SingleThreadModel` zaručí, že bude v jednu chvíli pouze jedno vlákno, které přistupuje k instančním proměným servletu.

```
public class YourServlet extends HttpServlet implements SingleThreadModel {
   // kód třídy
}
```

8.11.4. Metoda destroy

V případě, že je potřeba instanci servletu vypnout, pak ještě před tím, než se servlet ukončí se zavolá metoda `destroy`. Zde lze ukončit databázová spojení, zastavit procesy pozadí apod.

8.12. Formulářová data

Formuláře se používají pro předávání informací od prohlížeče k serveru. Odesílají se na server metodou GET nebo POST. Čtení formulářových dat na straně servletu je automatické. Servlet se postará o zjištění použité metody, dekóduje URL a rozpársuje parametry. K získání těchto parametrů slouží metoda následující metoda.

```
poblic String getParameter("name")
```

Této metoda vrací obsah daného parametru, prázdný řetězec v případě, že hodnota není vyplněná nebo null, v případě, že parametr neexistuje. Pokud by parametr mohl mít potenciálně více hodnot, pak použijeme metodu

```
public String[] getParameterValues("name")
```

Tato metoda vrací seznam hodnot. V případě že parametr neexistuje, vrací null. Další metoda vrátí seznam jmen parametrů jako `Enumeration`, jehož položky lze přetypovat na `String`.

```
public Enumeration getParameterNames()
```

8.13. Sledování sezení

Pro přenos dat po síti se využívá protokol http. Tento protokol je bezstavový, tzn., že server si neuchovává informace o jednotlivých požadavcích. Například internetový obchod využívající košík, potřebuje informace po celou dobu sezení uživatele o stavu košíku. Jinými slovy, server musí znát stav košíku daného uživatele u každého nového požadavku.

Tento problém se řeší třemi způsoby – Cookies, přepis URL a skrytá formulářová data.

8.13.1. API pro sledování sezení

Servlety používají pro sledování sezení třídu `HttpSession`. Jedná se o řešení vysoké úrovně, které využívá jak Cookies tak přepis URL automaticky.

Třída `HttpSession` definuje API, které se stará o vyhledávání objektu sezení sdruženého s aktuálním požadavkem, v případě nutnosti vytvoření nového sezení, uložení a zrušení.

Následující konstrukce najde objekt sezení pro daný požadavek. Systém na pozadí extrahuje identifikátor uživatele z cookie nebo z připojených dat URL. Ten pak použije jako klíč do tabulky již vytvořených objektů `HttpSession`. Pro programátora to je provedeno transparentně, vyvolá pouze metodu `getSession`. Pokud objekt neexistuje, uvedeným parametrem `true` se vytvoří nový. Pro zjištění zda objekt existoval se použije metoda `isNew`.

```
HttpSession session = request.getSession(true)
```

8.13.1.1. Metody HttpSession

```
public Object getAttribute(String name)
```

Metoda vrátí hodnotu atributu předanou jako parametr.

```
public void  setAttribute(String name, Object value)
```

Metoda nastaví atribut name a předá mu hodnotu value.

```
public void removeAttribute(String name)
```

Metoda odstraní atribut z objektu HttpSession.

```
public Enumeration getAttributeNames()
```

Metoda vrátí názvy všech atributů v sezení.

```
public String getId()
```

Vrací jedinečný identifikátor vytvořený pro každé sezení.

```
public boolean isNew()
```

Metoda vrátí true v případě, že sezení bylo právě vytvořeno.

```
public void invalidate()
```

Metoda ruší platnost sezení a uvolňuje všechny objekty s ním sdružené.

Ke zpracování kapitoly byla použita následující literatura [1], [2], [11], [12].

9. MWArray

MWArray je abstraktní třída, která slouží pro ruční převod datových typů z Javy do Matlabu a zpět. Při volání matlabovských funkcí z Java komponent, se musí předávané parametry převést na objektové typy Matlabu. K tomuto účelu Matlab Builder JA poskytuje API, které je implementováno abstraktní třídou `MWArray` jako

`com.mathworks.toolbox.javabuilder.MWArray`

balík. Tato abstraktní třída poskytuje několik podtříd. Každá podtřída reprezentuje jeden datový typ Matlabu. Jedná se o tyto třídy reprezentující hlavní datové typy Matlabu:

- MWNumericArray,
- MWLogicalArray,
- MWCharArray,
- MWCellArray a
- MWStructArray.

Třída `MWArray` a třídy odvozené z ní poskytují následující:

- konstruktory a finalyzéry pro vytvoření a zrušení MATLAB polí,
- metody `get` a `set` pro čtení a nastavování dat v poli,
- metody k určení vlastností polí,
- srovnání metod pro testování rovnosti nebo pořadí polí,
- metody pro převod do jiných datových typů.

Předáváme-li Matlabu z Java komponenty parametr typu `java.lang.Double`, musíme jej přetypovat na `MWNumericArray` následujícím postupem.

```
MWNumericArray dims = null;
dims = new MWNumericArray(Double.valueOf(args[0]),
             MWClassID.DOUBLE);
```

Následně můžeme parametr `dims` použít jako parametr při volání Matlabovské funkce.

```
result = theMagic.makesqr(1, dims);
```

MATLAB Builder JA převádí `MWNumericArray` na Matlab double proměnnou a předá ji Matlabovské funkci. Tyto převody provádíme buď ručně, anebo se tyto převody provádějí automaticky.

9.1. Pravidla pro přetypování parametrů

Pravidla pro přetypování java datových typů na datové typy Matlabu jsou uvedeny v tabulce 3 v příloze na CD `priloha/tabulky.pdf`.

Pravidla pro přetypování Matlab datových typů na java datové typy Matlabu jsou uvedeny v tabulce 4 v příloze na CD `priloha/tabulky.pdf`.

9.2. Generované rozhraní pro Matlab funkce

Matlab Builder JA generuje rozhraní jednotlivých Java metod tak, aby programovací jazyk Java podporoval volitelné argumenty funkce stejně jako MATLAB umožňuje práci s argumenty `varargin` a `varargout`.

Pro podporu této funkce builder generuje přetížené Java metody, které pojmou libovolný počet argumentů.

9.2.1. Matlab funkce

Obecná funkce matlabu má následující tvar

```
function [out1, out2, ..., varargout]=foo(in1, in2, ..., varargin)
```

Nalevo od rovnítka funkce specifikuje sadu explicitních a volitelných výstupních argumentů. Napravo od rovnítka jsou specifikovány explicitní vstupní argumenty následovanými jedním nebo více volitelnými argumenty.

Každý argument představuje datový typ Matlabu. Pokud jsou `varargin` a `varargout` uvedeny, můžeme zadat libovolný počet vstupů a výstupů za ty, které jsou explicitně deklarovány.

9.2.2. Přetížené java metody

Když Matlab Buidler JA zapouzdřuje matlab kód, tak vytváří přetížené metody, které implementují matlab funkce.

Kromě zapouzdření vstupních argumentů builder vytváří jinou metodu, která představuje generování proměnlivého počtu výstupních parametrů.

9.3. Vracení dat z Matlabu do Javy

Všechna data vrácena z matlabovských funkcí jsou představována jako instance příslušné podtřídy `MWArray`. Například buňka Matlab pole je vrácena jako objekt `MWCellArray`.

Návratová data nejsou převedena do Javy. Pokud budeme potřebovat data převést na Javovské datové typy, pak musíme zavolat metodu `toArray` podtřídy `MWArray`, která vrátí javovský typ pole.

9.4. Předávání argumentů do Javy a z Javy

Při volání metody Matlabu umístěné v Matlab Builder JA komponentě, musí být vstupní argumenty obdržené metodou v prostředí Matlab ve formátu pole. Tuto konverzi můžeme provést sami, nebo předat jako datový typ Java a nechat provést konverzi automaticky.

Pokud chceme převod provést sami, použijeme instance třídy `MWArray`. Toto je ruční konverze. Předáme-li javovské datové typy, pak spoléháme na automatický převod.

9.4.1. Ruční konverze

V následujícím kódu převedeme datový typ `java.lang.Double` na typ `MWNumericArray`, který lze poté použít v Matlab Builder JA komponnetě bez dalších konverzí. Následující kód využívá `MWNumericArray`.

```
MWNumericArray dims = null;
dims = new MWNumericArray(Double.valueOf(args[0]),
              MWClassID.DOUBLE);

result = theMagic.makesqr(1, dims);
```

Matlab Builder JA převádí objekt `MWNumericArray` na skalární Matlab double typ, který je pak předán funkci Matlabu.

9.4.2. Automatická konverze

Automatický převod z Javy do Matlabu umožňuje například převod primitivního datového typu nebo instance třídy `java.lang.Double` na datový typ Matlabu. Pravidla převodů jsou sepsána v tabulkách 3 a 4, které jsou umístěny v příloze na CD `priloha/tabulky.pdf`.

V následujícím výpise je volána metoda `myprimes`, kterí přijímá dva parametry. Prvním je očekávaný počet výstupů a druhým je objekt typu `java.lang.Double`, který bude automaticky převeden na datový typ pole Matlabu.

```
cls = new myclass();
y = cls.myprimes(1, new Double((double)n));
```

9.4.3. Změna defaultního typu

Pokud uvedeme typ požadovaných dat při volání konstruktoru `MWArray`, pak Matlab Builder JA převede argument na tento typ namísto původního.

V následujícím kódu máme proměnnou typu double, kterou převedeme na datový typ Matlabu INT16 pole namísto defaultního datového typu double pole.

```
double Adata = 24;
MWNumericArray A = new MWNumericArray(Adata, MWClassID.INT16);
System.out.println("Array A is of type " + A.classID());
```

9.5. Příklady použití MWArray

V této kapitole jsou umístěny praktické zkušenosti získané při práci s třídou `MWArray`. Jsou zde popsány některé užitečné metody třídy `MWArray` a dále pak konstrukce pro předávání a získávání argumentů při práci s Java komponentou.

9.5.1. Použití metod MWArray

Metoda `getClass()` vrací název datového typu objektu. Vrací-li funkce komponenty číslo reprezentované třídou `MWNumericArray`, pak následující kód:

```
Object[] out = myComponent.myFunction(1);
Systém.out.println(out[0].getClass());
```

vypíše

```
Class com.mathworks.toolbox.javabuilder.MWNumericArray
```

Vrací-li komponenta pole čísel, pak jsou tyto čísla reprezentována opět datovým typem `MWNumericArray`.

```
Object[] out = myComponent.myFunction(1);
```

Jednotlivé prvky získáme voláním metody `getDouble(int i)`, kde `i` je index prvku. První prvek má index 1.

```
out[0].getDouble(1);    // vrátí první prvek pole
out[0].getDouble(2);    // vrátí druhý prvek pole
```

V případě, že komponenta vrací dvě pole, pak bychom třetí prvek druhého pole získali následovně.

```
Object[] out = myComponent.myFunction(2);
out[1].getDouble(3);    // vrátí třetí prvek z druhého pole.
```

Další užitečnou funkcí je funkce `getDimensions()`. Tato metoda vrací dvouprvkové pole, které udává počat řádků a počet sloupců v objektu, nad kterým je tato metoda zavolána.

```
out.getDimensions()[0];       // vrátí počet řádků
out.getDimensions()[1];       // vrátí počet sloupců
```

Funkce `getDouble()` vrací jednu hodnotu na určitém indexu. Pokud je potřeba získat všechny hodnoty pole, využijeme metodu `getDoubleData()`, která vrací pole hodnot typu double.

```
MWNumericArray mwna = (MWNumericArray)out[0];
double[] data = mwna.getDoubleData();
data[1];                // vrátí druhý prvek
out[0].getDouble(2);    // vrátí druhý prvek
```

9.5.2. Předávání argumentů z Javy do Matlabu

9.5.2.1. Předání double pole

Předání pole s hodnotami datového typu double je velmi jednoduché. V tomto případě lze využít automatické konverze.

```
Double[] data = {1.1, 1.3, 1.0};
Object[] out = myComponent.myFunction(1, data);
```

9.5.2.2. Předání dvourozměrného pole

Předání dvourozměrného pole typu double je rovněž jednoduché a lze využít automatické konverze.

```
double[][] data = {{1.1, 1.3}, {2.1, 2.3}};
Object[] out = myComponent.myFunction(1, data);
```

9.5.3. Vracení argumentů z Matlabu do Javy

9.5.3.1. Získání čísla double a pole typu double

Vrací-li komponenta double číslo, je v Javě reprezentováno jako objekt MWNumericArray. Pokud s touto hodnotou budeme chtít pracovat, je potřeba toto číslo převést na javovský datový typ double následovně.

```
Object[] out = myComponent.myFunction(1);
out[0].getClass();      // datový typ MWNumericArray
MWNumericArray mwna = new MWNumericArray(out[0]));
Double d = mwna.getDouble(1);
```

S polem je práce stejná až na jeden příkaz a to poslední.

```
double[] d = mwna.getDoubleData();
```

9.5.3.2. Získání dvourozměrného pole

Máme-li Matlab funkci, která vrací dvourozměrné pole. Pak v Javě bude reprezentováno jednorozměrným polem s datovým typem `MWNumericArray`. Pomocí metody

`getDimensions()` získáme rozměr původní matice a pomocí metody `getDoubleData()` získáme prvky matice.

```
Object[] out = myComponent.myFunction(1);
int[] dim = out[0].getDimensions();
double[] vektor = ((MWNumericArray[])out[0]).getDoubleData();
```

Dvourozměrné pole lze sestavit následující konstrukcí.

```
double[][] matice  = new double[dim[0]][dim[1]];
for(int i = 0; I < dim[0]; i++) {
   for(int j = 0; j < dim[1]; j++) {
      double[i][j] = vektor[(j*dim[0]+i)]
   }
}
```

10. WebFigure

Nejčastějšími výsledky při výpočtech v Matlabu jsou grafy nebo animace. Grafy lépe znázorňují výsledky než čísla a v praxi jsou více používané při práci se signály. Proto se tato kapitola věnuje vytvářením grafiky v Matlabu , přenos k webovému prohlížeči a následnému zobrazení.

Jako uživatelské rozhraní pro spouštění Java komponent lze použít jak klasickou stránku s HTML formulářem, tak i applet.

10.1. Vytvoření grafiky v Matlabu

Mějme MATLAB funkci, která zobrazí sinusový graf. MATLAB využívá k vykreslení grafických objektů objekt figure. Pro zobrazení na webové stránce využijeme objekt webfigure, do kterého umístíme figure.

```
function [w_fig] = sinus()
A = 1; f = 4;
t = 0:0.01:1;
x = A*sin(2*pi*f*t);
h_fig = figure('visible', 'off', 'Menubar', 'none', 'Name', 'Sinus graf');
w_fig = webfigure(h_fig);
close(h_fig);
```

10.2. Zobrazení grafiky HTML souborem

Tento přístup představuje zobrazení grafiky pomocí HTML tagu, který se odkazuje na servlet, který generuje HTML kód, jež se postará o zobrazení grafiky ve webovém prohlížeči.

Servlet nejprve spustí metodu Java komponenty a získá z ní výsledky. Grafiku převede na objekt typu `WebFigure`. Získá se aktuální sezení metodou `getSession()`, do kterého se vloží grafika pod názvem atributu `sinus_webfigure`. Následně se zpět uživateli vygeneruje výstupní stránka a odešle nazpět prohlížeči. Grafika se do výstupní stránky vloží pomocí metody `getHTMLEmbdedString()`, která vytvoří HTML kód, který se postará o zobrazení grafiky ve výstupní HTML stránce. HTML kód generuje třída `WebFigures` z balíčku `javabuilder.jar`.

Výpis hlavní části servletu je na následujícím výpisu.

```
Object[] out = myComponent.sinus(1);
WebFigure wb = (WebFigure)MWJavaObjectRef.unwrapJavaObjectRefs(out[0]);
request.getSession().setAttribute("sinus_webfigure", wb);

response.setContentType("text/html;charset=UTF-8");
WebFigures webFigures = new WebFigures("WebFigures", getServletContext());
PrintWriter print = response.getWriter();
...
out.println(webFigures.getHtmlEmbdedString(wb, "sinus_webfigure",
  "session", "", "", ""));
...
```

Výstupní HTML stránka obsahuje následující kód.

```
...
<iframe src="/webfigure/Webfigures/getGraf?name=sinus_webfigure&
  scope=session">
</iframe>
...
```

o HTML stránky se vloží iframe, do kterého se nahraje výstup `servletu`. Konkrétní třída, která se stará o zobrazování grafiky je

`com.mathworks.toolbox.javabuilder.webfigures.WebFiguresServlet`.

10.3. Zobrazení grafiky appletem

Je-li použit applet pro zadání vstupních parametrů ke spuštění Java komponenty, pak se použije jiná konstrukce pro zobrazení grafiky. Applet vytvoří požadavek GET nebo POST a odešle na servlet. Servlet spustí Java komponentu a získanou grafiku odešle zpět appletu jako binární data. Applet pak zobrazí obrázek. Kód servletu se trošku změní následovně.

```
import java.awt.image.BufferedImage;
import java.imageio.ImageIO;
...
Object[] out = myComponent.sinus(1);
BufferedImage bufferedImage =
  Images.renderArrayData((MWNumericArray)out[0])
request.getSession().setAttribute("sinus_webfigure", bufferedImage);
...
response.setContentType("image/jpeg");
ImageIO.write(bufferedImage, "jpg", response.getOutputStream());
...
```

V appletu vytvoříme objekt URL, který reprezentuje url souboru servletu společně s předávanými argumenty. Následně se odešle požadavek na výstup daného souboru a umístí se na požadované místo. Obsah proměnné target odkazuje na iframe s názvem content_appl.

```
URL servletURL  = new URL(stringURL);
String target = "content_appl";
getAppletContext().showDocument(servletURL, target);
```

11. Webová aplikace využívající Java komponentu

Tento příklad ukazuje zobrazení grafu vytvořeného Java servletem, který zavolá komponentu vytvořenou pomocí Java Builder JA. Zdrojové kódy jsou umístěny v příloze na CD v adresáři `aplikace/Signal`.

11.1. Požadavky

Pro nasazení webové aplikace využívající Java komponentu potřebujeme mít nainstalovaný aplikační server podporující J2EE. Dále je potřeba nainstalovat `javabuilder.jar` a `servlet-api.jar` do adresáře společných knihoven. V příloze na CD jsou zdrojové kódy webové aplikace, která zobrazí graf sinusového signálu. Aplikace se nachází v adresáři priloha/`aplikace/Signal`. V tomto adresáři jsou umístěny zdrojové kódy:

- `matlab/getSignal.m` – matlab kód zobrazující graf,
- `matlab/distrib/Signal.jar` – Java komponenta,
- `servlet/src/SignalServlet/SignalServletClass.java` – servlet připravený ke kompilaci,
- `servlet/webfigures.tld` – soubor potřebný pro používání webfigure v servletech,
- `servlet/web.xml` – konfigurační soubor servletu,
- `lib` – adresář se soubory `javabuilder.jar` a `servlet-api.jar`,
- `kompilace.bat` – dávkový soubor pro kompilaci servletu,
- `html/index.html` – html soubor pro spuštění Java komponenty a zobrazení grafu.

11.2. Postup tvorby webové aplikace

Následuje návod na sestavení Java komponenty, napsání servletu, nasazení servletu na aplikační server a spuštění pro ověření správné funkčnosti.

- Spustíme nástroj ***deploytool*** a vytvoříme nový projekt `Signal.prj` v adresáři `matlab`.
- Přidáme soubor `getsignal.m`.
- Java komponentu sestavíme stisknutím tlačítka `Build the project`.
- Spustíme soubor `kompilace.bat` z kořenového adresáře aplikace. Obsah souboru `kompilace.bat` je

```
javac -classpath .\lib\javabuilder.jar;.\lib\servlet-api.jar;
   .\matlab\Signal\distrib\Signal.jar
   .\servlet\src\SignalServlet\SignalServletClass.java
```

- Tímto se vytvoří class soubor `SignalServletClass.class`.
- V adresáři `servlet\build` vytvoříme následující adresářovou strukturu:

- do adresáře `WEB-INF` vložíme soubory `web.xml` a `webfigures.tld`.
- do adresáře `WEB-INF\lib` vložíme soubory `javabuilder.jar`, `servlet-api.jar` a `Signal.jar`.
- do adresáře `WEB-INF\classes\SignalServlet` vložíme soubor `SignalServletClass.class`.
- Do kořenového adresáře `servlet\build` vložíme soubor `index.html`.

- Vytvoříme archiv WAR následujícím příkazem

```
jar cf Signal.war –C build .
```

- WAR archiv `Signal.war` nakopírujeme do adresáře `tomcat/webapps` a tomcat restartujeme.
- Aplikace je dostupná na URL `http://localhost:8080/Signal/`.

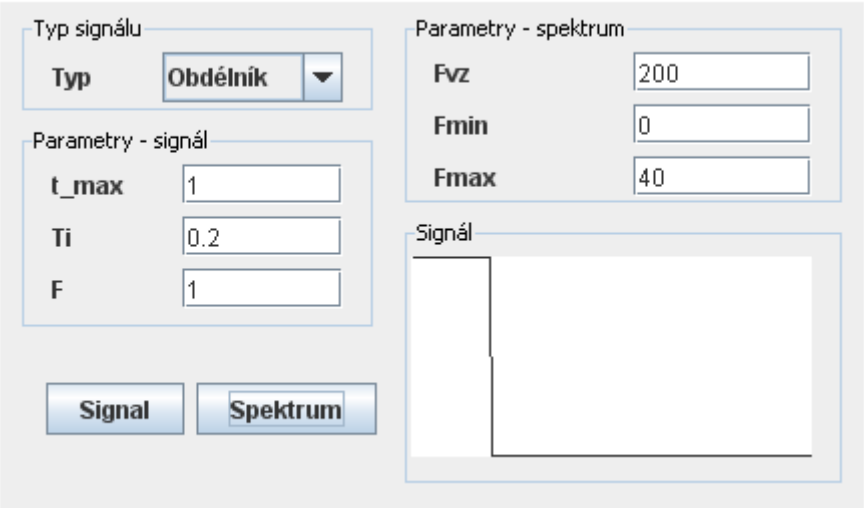
12. Ukázkové aplikace

V rámci diplomové práce jsou vytvořené dvě aplikace. První z nich je aplikace pro výpočet amplitudového spektra signálu a druhou aplikací je simulace modulací.

12.1. Aplikace – Amplitudové spektrum

V aplikaci lze vygenerovat následující signály: sinusový, trojúhelníkový, obdélníkový a pilovitý. Uživatelské rozhraní aplikace je naprogramováno v Javě jako applet. V appletu je umístěno kreslící plátno, ve kterém se vygenerovaný signál zobrazí a je možné v něm také tento signál upravovat tahem myší. Výstupní grafika s vypočteným amplitudovým spektrem se zobrazí jako obrázek ve webovém prohlížeči.

Na následujícím obrázku je zobrazeno uživatelské rozhraní appletu. Applet je rozdělen do čtyř částí.

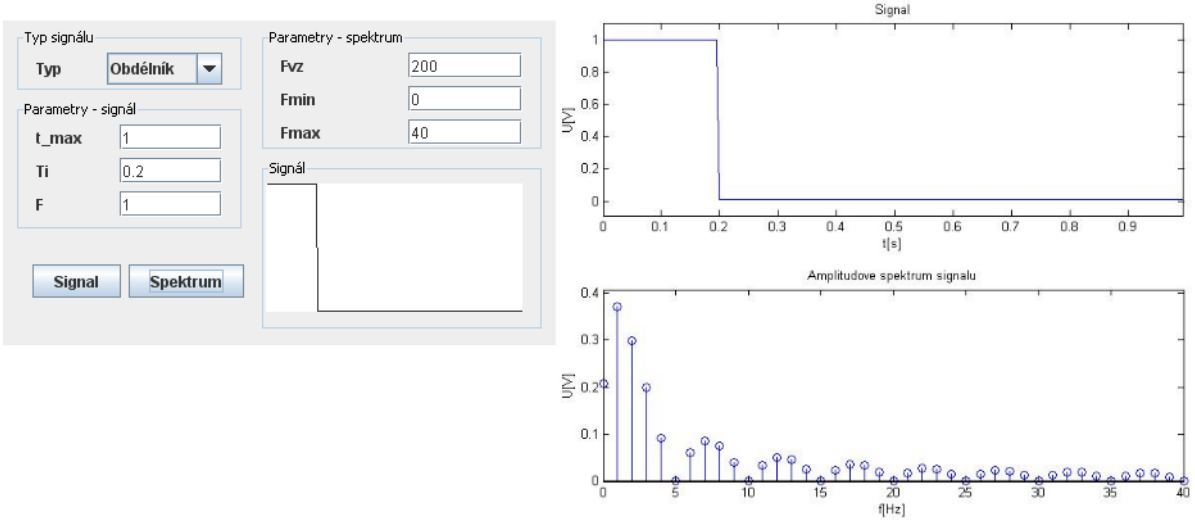


Obr. 6: Applet – Spektrum signálu

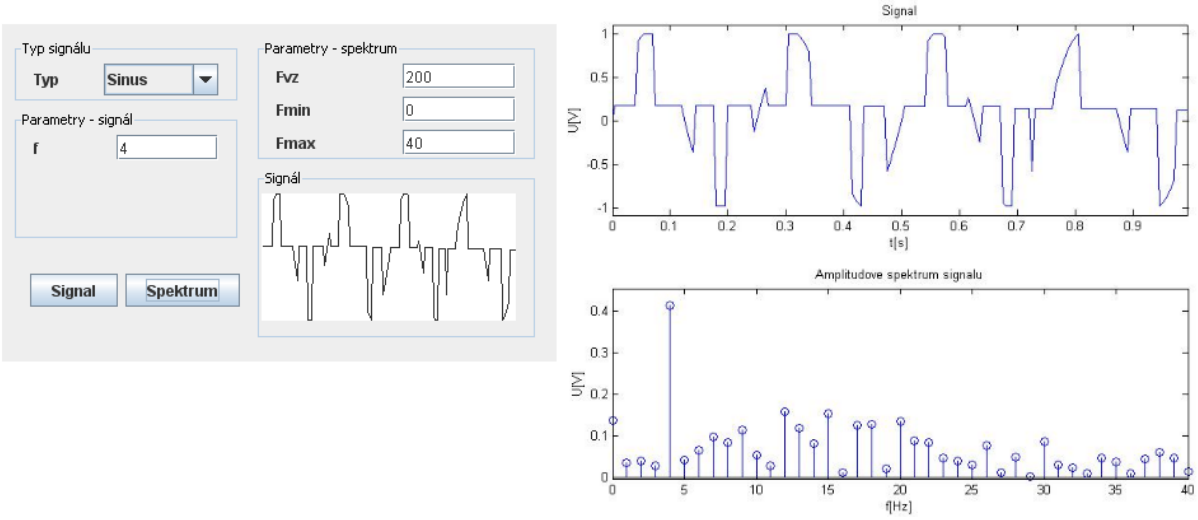
V panelu ***Typ signálu*** se vybírá typ signálu. Ve druhém panelu ***Parametry – signal*** se zadávají parametry signálu. Konkrétně pro obdélníkový signál se zadává `t_max`, jež určuje délku jedné periody. V tomto případě hodnota 1 odpovídá trvání 1 s. Parametr `Ti` určuje délku impulzu a `F` uvádí počet period. Ve třetím panelu ***Parametry – spektrum*** se zadává frekvence vzorkování `Fvz`. Původní signál v kreslícím plátnu má 200 vzorků. Pro potřeby navýšení počtu vzorku slouží právě tento parametr. Parametr `Fmin` a `Fmax` udávají rozsah kmitočtů, které se zobrazí v amplitudovém spektru. Ve čtvrtém panelu ***Signal*** je umístěno kreslící plátno, ve kterém se zobrazuje průběh signálu. Tento tvar signálu lze myší překreslovat. Tlačítkem

Signal provedeme aktualizaci signálu podle zadaných parametrů. Tlačítkem ***Spektrum*** se provede výpočet a zobrazí se grafy.

Následují dvě ukázky výpočtu amplitudového spektra obdélníkového signálu a signálu nakresleného rukou.



Obr. 7: Applet – spektrum signálu (příklad 1)



Obr. 8: Applet – spektrum signálu (příklad 2)

Zdrojové kódy všech komponent aplikace (matlab, servlet, applet) jsou v příloze na CD v adresáři `aplikace/SpectrumSignal`.

12.2. Simulace modulací

Tato aplikace umožňuje simulovat základní analogové a digitální modulace. Uživatelské rozhraní je rovněž napsáno jako applet, který využívá kreslící plátno pro vizualizaci a úpravu modulačního signálu. Java komponenta implementuje následující analogové modulace, jejichž zdrojové kódy byly přejaty z mé bakalářské práce [13]:

- amplitudová modulace (AM),
- amplitudová modulace se dvěma postranními pásmy s redukovanou nosnou (AM-DSB-SC),
- amplitudová modulace s jedním postranním pásmem (AM-SSB),
- frekvenční modulace (FM),
- fázová modulace (PM),
- pulzně amplitudová modulace (PAM),
- pulzně polohová modulace (PPM),
- pulzně šířková modulace (PWM)

a dále implementuje digitální modulace, se kterými lze modulovat digitální signál vytvořený z binární posloupnosti:

- modulace s amplitudovým klíčováním (ASK),
- modulace s frekvenčním klíčováním (FSK) a
- modulace s fázovým klíčováním (PSK).

Aplikace generuje 4 grafy:

- průběh modulačního signálu,
- průběh modulovaného signálu,
- amplitudové spektrum modulačního signálu a
- amplitudové spektrum modulovaného signálu.

Applet obsahuje 5 panelů a tlačítka.

Panel ***Modulační signál*** umožňuje vybrat požadovaný typ signálu a nadefinovat jeho parametry.
U sinusového signálu lze zadat frekvenci vzorkování a kmitočet signálu.

Panel ***Modulace*** umožňuje vybrat typ modulace. Konkrétně u amplitudové modulace se nastavuje parametr kmitočet nosné vlny a hloubka modulace.

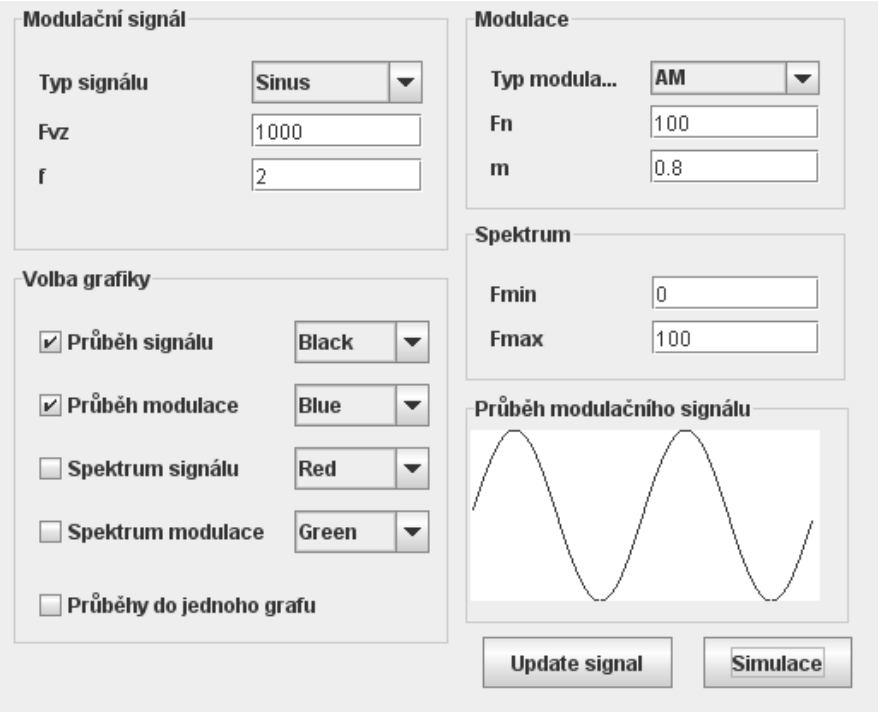
V panelu ***Spektrum*** se nastavuje rozsah kmitočtů, které mají být zobrazeny v amplitudovém spektru.

Panel ***Volba grafiky*** umožňuje zvolit požadované grafy a přiřadit jim určitou barvu. Poslední kolonka v tomto panelu sloučí modulační signál a modulovaný signál do jednoho grafu.

Poslední panel ***Průběh modulačního signálu*** využívá kreslící plátno, které zobrazuje aktuální tvar modulačního signálu. Tento tvar lze myší libovolně měnit.

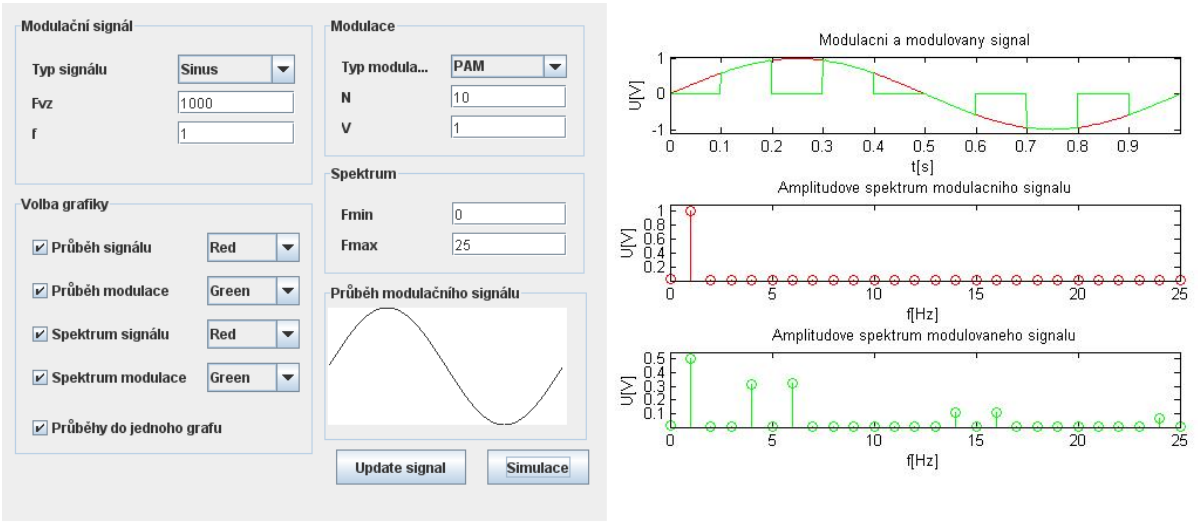
Výstupem takového appletu je následující obrázek.

Tlačítkem ***Update signal*** lze dát tvar signálu do původního stavu před kreslením do kreslícího plátna. Tlačítkem ***Simulace*** se získá požadovaná grafika. Applet je na následujícím obrázku.

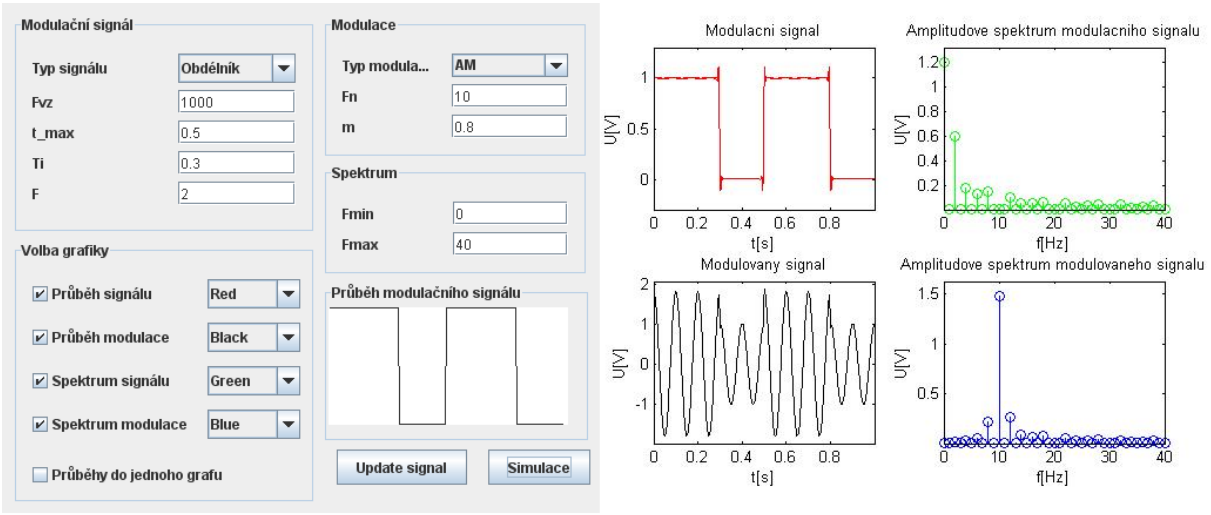


Obr. 9: Applet - Modulace

Na dalších obrázcích jsou ukázky aplikace simulace modulací.



Obr. 10: Applet – Modulace (příklad 1)



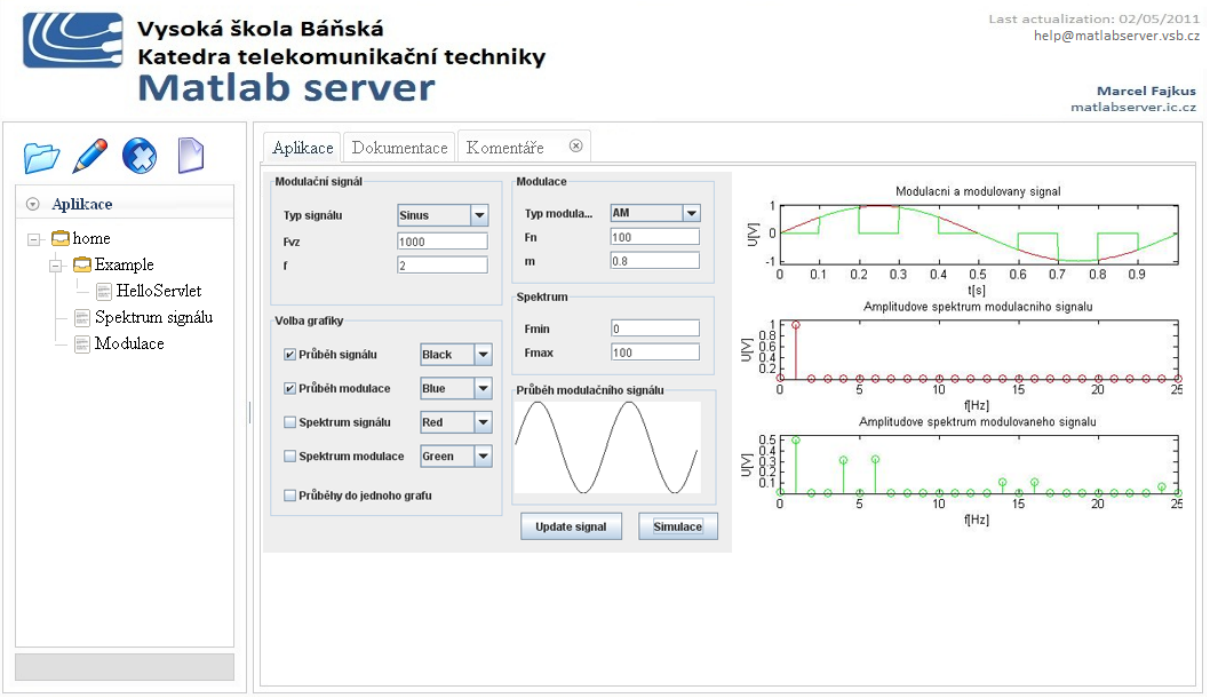
Obr. 11: Applet – Modulace (příklad 2)

Zdrojové kódy jsou umístěny v příloze na CD v adresáři `aplikace/Modulace`.

12.3. Webová stránka MATLAB serveru

Pro katderu jsem navrhl a implementoval webovou stránku MATLAB serveru, na které lze přistupovat k jednotlivým aplikací. Webová stránka je naprogramována v jazyce HTML, PHP a CSS, využívající javascript, javascriptový framework Dojo, technologii Ajax a databázi mySQL pro ukládání dat.

Webová stránka umožňuje aplikace nahrávat, spouštět a mazat. Ukázka vzhledu webové stránky MATLAB serveru je na následujícím obrázku.



Obr. 12: Návrh webové stránky MATLAB serveru

13. Závěr

Diplomová práce byla zadána s požadavkem vytvořit Matlab server, jenž by byl využit na katedře telekomunikační techniky Vysoké školy Báňské -Technické univerzity Ostrava.

V práci jsou uvedeny požadavky tvorby Matlab aplikací pro nasazení na webový server, dále postupy vývoje těchto aplikací s jednoduchými příklady na vysvětlení a návody pro nasazení aplikace na server. V závěru jsou vytvořeny dvě ukázkové aplikace, které využívají applet jako uživatelské rozhraní aplikace pro zadání vstupních parametrů, dále servlety, jež na straně serveru přijímají požadavky a na základě těchto požadavků spouštějí vytvořenou Java komponentu obalující Matlab aplikaci. První z nich je výpočet a zobrazení amplitudového spektra vybraného signálu a druhou aplikací je simulace modulací, která navazuje na teoretickou část mé bakalářské práce [13] na téma *Modulace signálu a jejich vliv na spektrum signálu.*

Výsledkem diplomové práce je, kromě již výše zmíněného, nasazení Matlab serveru na školním serveru, zejména z pohledu aplikací vytvořenými pro studijní účely, podobně jako je tomu u aplikace simulace modulací.

Navíc je ještě vytvořena webová stránka `http://matlabserver.ic.cz` s obsahem této diplomové práce, návody a příklady aplikací.

Seznam příloh na CD

Na CD jsou umístěny přílohy v adresáři `priloha`. Adresářová struktura příloh je následující

`aplikace/`	- adresář se zdrojovými kódy aplikací uvedených v DP
`aplikace/Modulace/`	
`aplikace/SpectrumSignal/`	
`aplikace/Signal/`	
`aplikace/Sumarum/`	
`aplikace/HelloServlet/`	
`tabulky.pdf`	- pdf soubor s tabulkami
`zdrojove_kody/`	- adresář s výpisy zdrojových kódů, které se nevešly do textu diplomové práce
`MCR/mcrinstaller.exe`	- Adresář s instalačním MCR použitým při vytváření Java komponent v této diplomové práci

Literatura

[1] HALL, Marty. *Java : servlety a stránky JSP*. [s.l.] : Neocortex, 2001. 586 s. ISBN 80-86330-06-0.

[2] KOBERNA, Lukáš. *Internetové rozhraní do Matlabu*. [s.l.], 2008. 52 s. Bakalářská práce. České vysoké učení technické v Praze.

[3] *Mathworks.com* [online]. 2011 [cit. 2011-05-02]. The Application Deployment Products and the Deployment Tool. Dostupné z WWW: <http://www.mathworks.com/help/toolbox/compiler/bsfey7f.html#bsfj5r8>.

[4] *Artax* [online]. 2011 [cit. 2011-05-02]. J2EE - Java 2 Enterprise Edition. Dostupné z WWW: <http://artax.karlin.mff.cuni.cz/~ebik/nju/linuxem/J2EE/J2EE.html>.

[5] *Mathworks.com* [online]. 2011 [cit. 2011-05-02]. Working with the MCR. Dostupné z WWW: <http://www.mathworks.com/help/toolbox/compiler/f12-999353.html#br2jauc-34>.

[6] *Mathworks.com* [online]. 2011 [cit. 2011-05-02]. Getting Started. Dostupné z WWW: <http://www.mathworks.com/help/toolbox/javabuilder/ug/brlu1as-1.html>.

[7] *The Apache Software Foundation* [online]. 2011 [cit. 2011-05-02]. The Apache Tomcat Connector - Reference Guide. Dostupné z WWW: <http://tomcat.apache.org/connectors-doc/reference/apache.html>.

[8] *The Apache Software Foundation* [online]. 2011 [cit. 2011-05-02]. The JK Connector. Dostupné z WWW: <http://tomcat.apache.org/tomcat-4.1-doc/config/jk.html>.

[9] AULDS, Charles. *Linux administrace serveru Apache*. [s.l.] : [s.n.], 2003. 536 s. ISBN 80-247-0640-7.

[10] KÁL, Peter. *Inštalácia aplikačného servera Tomcat, load balancing a porovnanie s aplikačným serverom Oracle*. [s.l.], 2008. 63 s. Bakalářská práce. Univerzita Pavla Jozefa Šafárika v Košiciach.

[11] PERRY, Bruce. *Java Servlet & JSP Cookbook*. [s.l.] : O, 2004. 752 s. ISBN 978-0-596-00572-6.

[12] MAIXNER, David. *Java servlety, JSP a webové servery s jejich podporou*. [s.l.], 2002. 42 s. Bakalářská práce. Masarykova univerzita.

[13] FAJKUS, Marcel. *Modulace signálu a jejich vliv na spektrum signálu*. [s.l.], 2009. 58 s. Bakalářská práce. VŠB – Technická Univerzita Ostrava. Dostupné z WWW: <http://www.modulace.ic.cz/download/bak_modulace.pdf>.